

PCIバス/ PCIExpress対応
モーションコントロールボード

MC8000P

デバイスドライバ

取扱説明書

初版	2006. 11. 16	[Ver 1.0.0.0]	
改訂	2017. 1. 11	[Ver 8.0.0.0]	
改訂	2017. 9. 12	[Ver 8.0.0.0]	(C#対応)
改訂	2018. 1. 30	[Ver 8.0.0.0]	(PCI Express対応)

対応ボード

MC8000Pシリーズ	MC8043P/ MC8043Pe
	MC8082P/ MC8082Pe
	MC8022P
	MC8042P
MC8500Pシリーズ	MC8541P/ MC8541Pe
	MC8581P/ MC8581Pe

はじめに

このたびは、ノヴァエレクトロニクス製PCIバス/PCI Express対応モーションコントロールボードをご検討いただきまして、ありがとうございます。

■安全にお使いいただくために

本製品を安全にお使いいただくために、本書に記述されている内容を必ずお守りください。
なお、注意事項をお守りいただかない場合、製品の故障、瑕疵担保責任、その他一切の保証をできかねる場合があります。
本製品をご使用いただく前に、必ず本書を熟読し理解した上でご使用ください。

また、本書の記載内容は、今後、機能の向上などのため予告なしに変更する場合があります。
最新の取扱説明書、ソフトウェアは、弊社ホームページ（URL: <http://www.novaelec.co.jp/>）からダウンロードできます。

■マニュアルの併用

ボードの仕様などについては、ハードウェア取扱説明書を参照してください。

■本書で使用する特殊用語

IC ICとはMCX314As、MCX304、MCX514の事を指します。

IC-A、IC-B ボードに複数ICを搭載している場合、1つ目のICを「IC-A」、2つ目のICを「IC-B」と表現します。ボードに搭載しているICが1つの場合、そのICを「IC-A」と表現します。

目次

1. 概要	1
1.1 対応ボード	1
1.2 対応OS	1
1.3 対応言語	1
1.4 最大ボード数	1
1.5 ボード搭載IC	1
2. インストール	2
2.1 ドライバソフトウェアの準備	2
2.2 本ボードを複数枚使用する場合の設定	2
2.3 パソコンへの本ボードの組込み	2
2.4 デバイスドライバのインストール	3
2.5 デバイスドライバのアンインストール	7
2.5.1 デバイスドライバのアンインストール	7
2.5.2 異なる種類のボードをインストールしている場合のアンインストール	11
2.6 デバイスドライバの更新	11
3. プログラミング	13
3.1 動作環境	13
3.2 ソフトウェア構成	13
3.2.1 ソフトウェア一覧	13
3.2.2 サンプルプログラムおよび評価ツールの実行時にエラーが発生する場合の対応	15
3.3 開発手順	16
3.3.1 VC++の場合	16
3.3.2 V.B.N.E.Tの場合	16
3.3.3 C#の場合	16
4. MC8000Pシリーズ ボード	18
4.1 API	18
4.1.1 関数一覧	18
4.1.2 関数仕様	21
4.1.3 補足説明	68
4.1.4 使用方法	81
4.2 プログラミング上の注意点	86
4.3 MCX304評価ツール	89
4.3.1 実行プログラムについて	89
4.3.2 機能概要	89
4.3.3 メイン画面	90
4.3.4 モード設定画面	91
4.3.5 自動原点出しモード設定画面	91
4.3.6 ステータス画面	92
4.3.7 Port A, B, C 出力画面	92
4.4 MCX314As評価ツール	93
4.4.1 実行プログラムについて	93
4.4.2 機能概要	93
4.4.3 メイン画面	94
4.4.4 モード設定画面	95
4.4.5 拡張モード設定画面	95
4.4.6 同期動作モード設定画面	95
4.4.7 ステータス画面	96
5. MC8500Pシリーズ ボード	97
5.1 API	97
5.1.1 関数一覧	97
5.1.2 関数仕様	100
5.1.3 補足説明	171
5.1.4 使用方法	186
5.2 プログラミング上の注意点	192
5.3 MCX514評価ツール	193
5.3.1 実行プログラムについて	193
5.3.2 機能概要	193

5.3.3	メイン画面	194
5.3.4	ライトレジスタ設定画面	195
5.3.5	同期動作設定画面	196
5.3.6	自動原点出し設定画面	196
5.3.7	PIO信号設定画面	197
5.3.8	多目的レジスタ/入力信号フィルタ設定画面	197
5.3.9	補間モード設定画面	197
5.3.10	リードレジスタ画面	198

1. 概要

本デバイスドライバは、ノヴァエレクトロニクス製PCIバス/PCI Express対応モーションコントロールボードデバイスドライバです。ボードの仕様は、ボードの取扱説明書とボードに搭載している I C の取扱説明書を参照して下さい。

1.1 対応ボード

本デバイスドライバは下記のボードに対応しています。1つのPCに複数の種類のボードをインストールできます。

対応ボード	
MC8000Pシリーズ	MC8043P / MC8043Pe
	MC8082P / MC8082Pe
	MC8022P
	MC8042P
MC8500Pシリーズ	MC8541P/MC8541Pe
	MC8581P/MC8581Pe

1.2 対応OS

本デバイスドライバは下記のOSに対応しています。

対応OS
Windows7(32bit, 64bit)
Windows8.1(32bit, 64bit)
Windows10(32bit, 64bit)

1.3 対応言語

本デバイスドライバは下記の言語に対応しています。
アプリケーションから弊社が提供するDLLを呼び出す事でボードを制御できます。

対応言語
Microsoft Visual C++
Microsoft Visual Basic
Micorsoft Visual C#

注) 対応しているMicrosoft Visual Studioのバージョンおよび、言語に対応する開発環境とOSとの関係はMicrosoftのホームページおよび当社ホームページを参照ください。

1.4 最大ボード数

本デバイスドライバは対応ボードを同時に16枚まで認識します。
1つのPCに複数の種類のボードをインストールした場合も最大16枚までです。

1.5 ボード搭載IC

対応ボードに搭載しているICは下記の通りです。

ボード	搭載IC	IC数	軸数
MC8043P / MC8043Pe	MCX314As	1	4
MC8082P / MC8082Pe	MCX304	2	8
MC8022P	MCX304	1	2
MC8042P	MCX304	1	4
MC8541P / MC8541Pe	MCX514	1	4
MC8581P / MC8581Pe	MCX514	2	8

2. インストール

この章では、本ボードのパソコンへの組込みとデバイスドライバのインストール方法について説明します。

2.1 ドライバソフトウェアの準備

CD-ROMからデバイスドライバをインストールする場合は、MC8000PデバイスドライバのCD-ROMを用意して下さい。ホームページからダウンロードしたデバイスドライバをインストールする場合は、ダウンロードしたソフトウェアを解凍して下さい。

2.2 本ボードを複数枚使用する場合の設定

本デバイスドライバは本ボードを同時に16枚まで認識します。

本ボードを1つのシステム(PC)で複数枚使用する時は、ボードを個別に認識させる為に、2枚目以降のボードはボード番号をボード上のロータリースイッチで設定して下さい。

ロータリースイッチ(SW1)の位置は、ボードの取扱説明書「基板外形」の章を参照してください。

ロータリースイッチは0～Fのいずれかを設定できます。ロータリースイッチの番号は他のボードと重複しないように設定して下さい。ボード番号は、異なる機種 of ボードにおいても重複しないで下さい。

出荷時は、ボード上のロータリースイッチに0が設定されています。

2.3 パソコンへの本ボードの組込み

注意：パソコンへの取り付け作業は必ずパソコンの電源を切断してから行ってください。さもないと回路素子を破壊する原因となります。

- ① パソコン本体の電源がOFFであることを確認してから、外装カバー、スロットカバー等を外します。
- ② 空いている拡張スロットへ本ボードを差し込みます。基板のエッジコネクタをパソコンのPCIバス/ PCI Expressコネクタに正しく挿入してください。
- ③ 取付金具をネジ止めしてください。この時キチンとねじを締めないと後で抜け落ちたりするなどして、ショートや故障、誤動作の原因となります。
- ④ パソコン本体の外装カバーを元通りに取り付けます。

2.4 デバイスドライバのインストール

この章では、本ボードのパソコンへの取り付けとデバイスドライバのインストール方法について説明します。

起動中の他のアプリケーションは終了させてください。

デバイスドライバのインストールは必ずアドミニストレーター権限をもったユーザーログインで行ってください。アドミニストレーター権限以外でインストールをした場合正常にインストールされません。

下記の手順はWindows 7(64bit)を例にしていますが、Windows 7(32bit)やWindows 8.1、Windows 10でもインストールの進め方は同じです。

- ① 2.1の方法でインストールするデバイスドライバを準備して下さい。
- ② 2.2, 2.3 を実行し、本ボードが確実にパソコンに取り付けられているか確認してください。
- ③ パソコン本体の電源をONし、Windows 7を起動します。
- ④ アドミニストレーター権限を持ったユーザーでログインしてください。
- ⑤ 提供CD-ROMのDriverフォルダ（提供CD-ROMがDドライブにある場合は、D:\Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、64 Driverフォルダ内の「64 install XXXX INF.exe」をダブルクリックしてください。（32bitの場合は32 Driverフォルダ内の「32 install XXXX INF.exe」をダブルクリックしてください。）
（XXXX にはお使いのボード名が入ります）



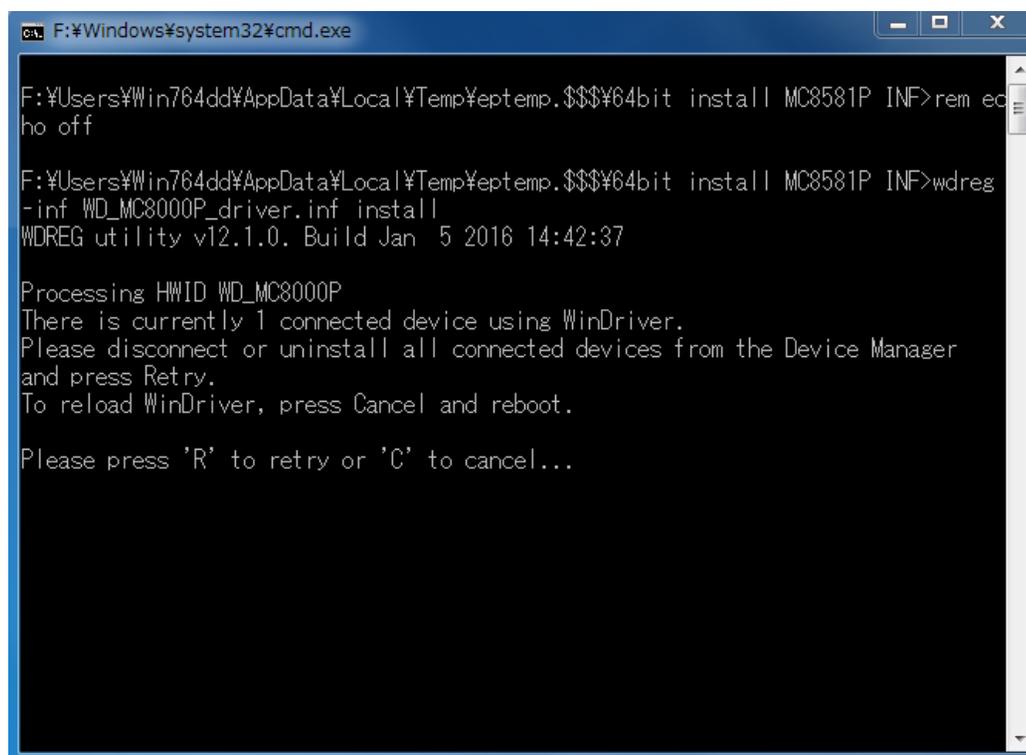
- ⑥ ”ユーザーアカウント制御” の画面が出てきた場合は、【はい】ボタンをクリックして操作を進めてください。（画面上のMC8541Pはお使いのボード名に置き換えてご覧下さい。）



- ⑦ デバイスドライバのインストール画面
【次へ】のボタンをクリック（画面上のMC8541Pはお使いのボード名に置き換えてご覧下さい。）



- ⑧ 次のような画面が、既に別のボードがインストールされている場合のみ表示されます。（初めてインストールする場合には表示されません。）
キーボードで[C]を入力します。

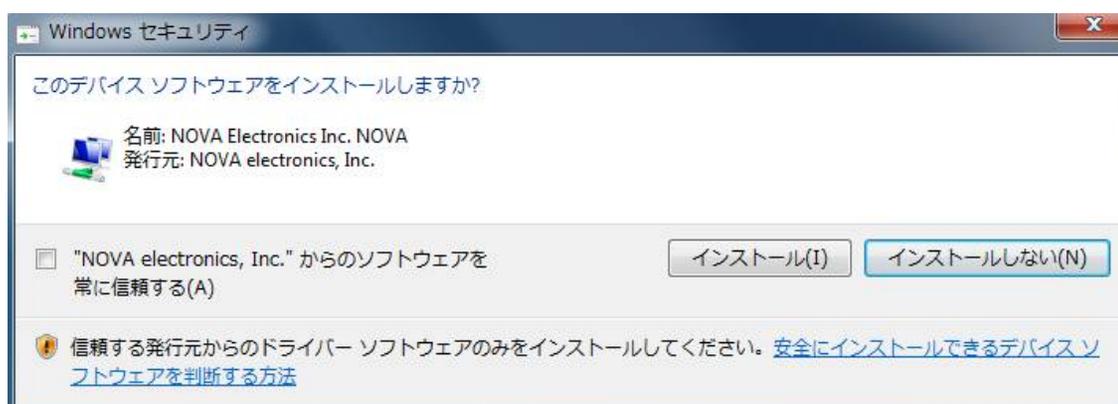


```
F:\Windows\system32\cmd.exe
F:\Users\Win764dd\AppData\Local\Temp\eptemp.$$$64bit install MC8581P INF>rem ed
ho off
F:\Users\Win764dd\AppData\Local\Temp\eptemp.$$$64bit install MC8581P INF>wdreg
-inf WD_MC8000P_driver.inf install
WDREG utility v12.1.0. Build Jan  5 2016 14:42:37

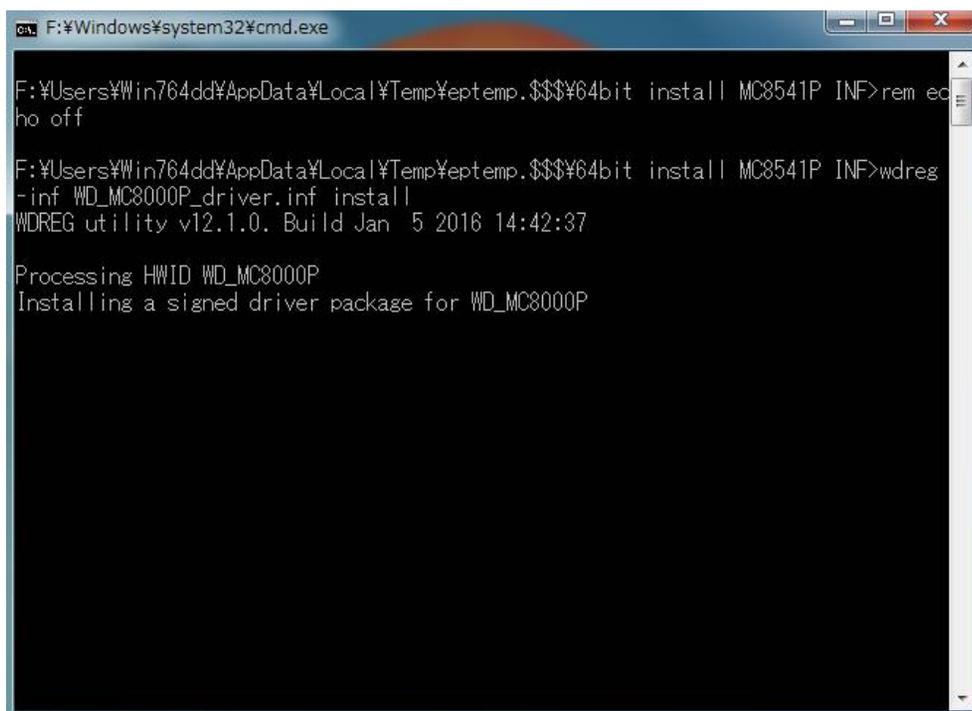
Processing HWID WD_MC8000P
There is currently 1 connected device using WinDriver.
Please disconnect or uninstall all connected devices from the Device Manager
and press Retry.
To reload WinDriver, press Cancel and reboot.

Please press 'R' to retry or 'C' to cancel...
```

- ⑨ 次のような画面が2つ表示されます。【インストール】をクリックします。



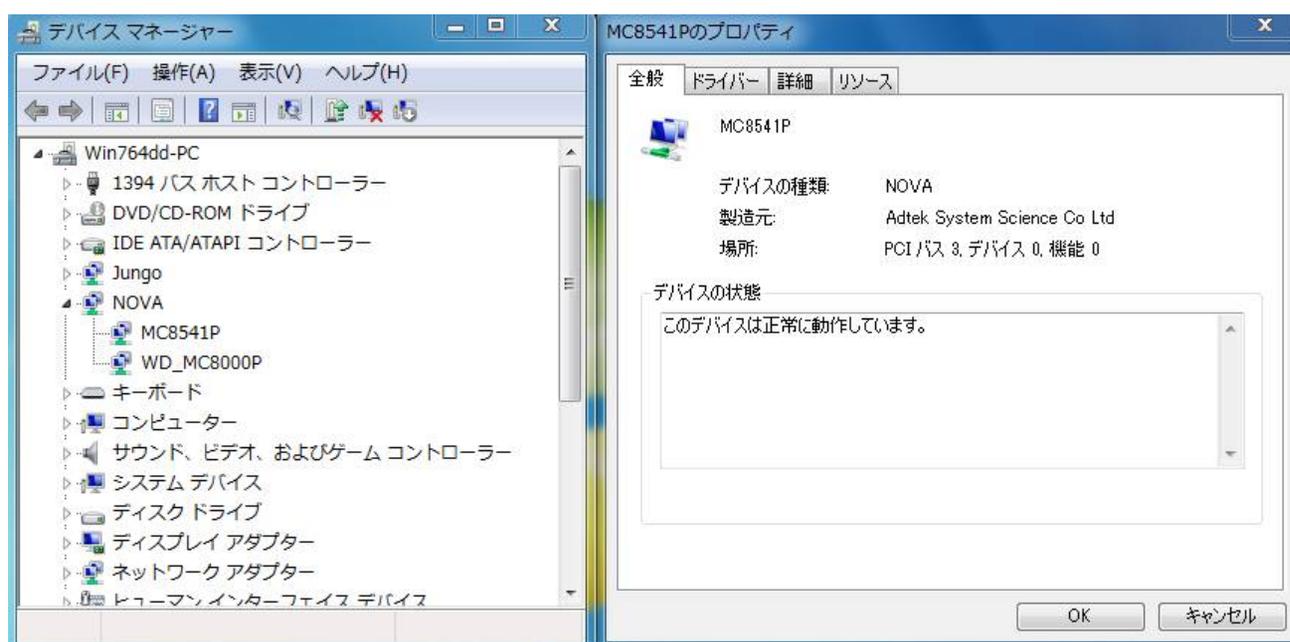
- ⑩ 次のような画面が表示され、処理が終了すると自動的に画面が閉じられます。(インストールに時間がかかることがあります。)



- ⑪ 【OK】 ボタンをクリックしてデバイスドライバのインストールは完了です。

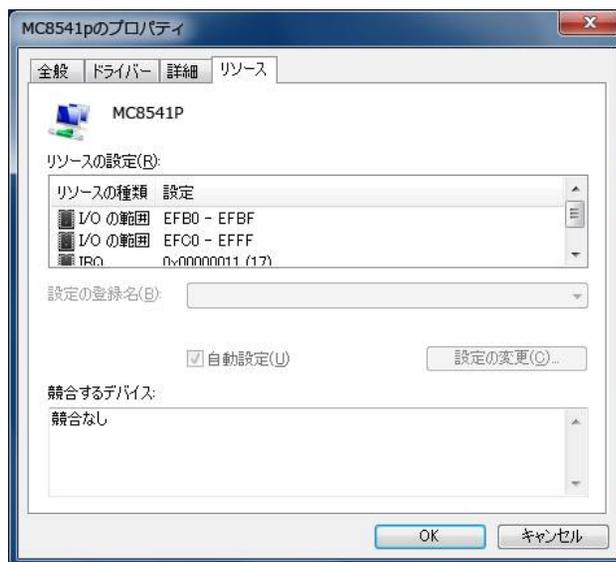


- ⑫ 以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システムとセキュリティ」－「デバイスマネージャ」画面（下記左画面）を開きます。
「NOVA」の下にボード名「MC XXXX P」と「WD_MC8000P」が表示されます。



ボード名をダブルクリックし、「全般」タブで上記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。
(画面上のMC8541Pはお使いのボード名に置き換えてご覧下さい。)
(PCI Express ボードでも、ボード名表示はPCI ボードと同じです。例えば、MC8082Pe をインストールした場合、ボード名表示は、MC8082P となります)

続いて、[リソース]タブで競合するデバイスに「競合なし」と記載されていることを確認してください。



2.5 デバイスドライバのアンインストール

この章では、本ボードのデバイスドライバのアンインストール方法について説明します。

下記の手順はWindows 7(64bit)を例にしていますが、Windows 7(32bit)やWindows 8.1、Windows 10でもアンインストールの進め方は同じです。

2.5.1 デバイスドライバのアンインストール

本ボードのデバイスドライバのアンインストール方法について説明します。

異なる種類のボードがインストールされている場合（MC8043PとMC8541Pの2種類のボードがインストールされている場合など）は「2.5.3 異なる種類のボードをインストールしている場合のアンインストール」を参照して作業を行ってください。

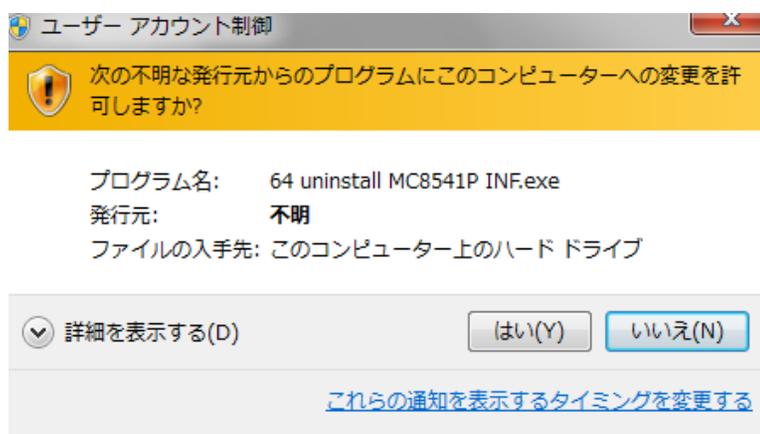
2.5.1.1 Uninstall XXXX INF.exeの実行

Uninstall XXXX INF.exe(XXXXにはお使いのボード名が入ります)を以下の手順で実行します。

- ① 提供CD-ROMのDriverフォルダ（提供CD-ROMがDドライブにある場合は、D:\Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、64 Driverフォルダ内の「64 uninstall XXXX INF.exe」をダブルクリックしてください。
Ver. 8.0.0.0以前のデバイスドライバを使用している場合には、Ver. 8.0.0.0以降のソフトウェアを使用して削除を行なってください。
(32bitの場合は32 Driverフォルダ内の「32 uninstall XXXX INF.exe」をダブルクリックしてください。)
(画面上のMC8541Pはお使いのボード名に置き換えてご覧ください。)



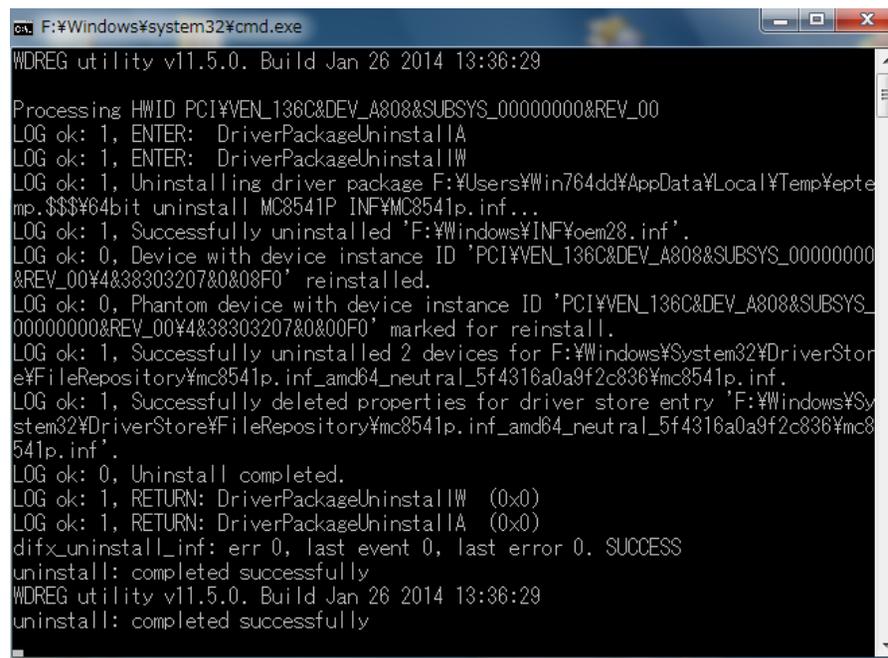
- ② "ユーザーアカウント制御" の画面が表示されたら、【はい】ボタンをクリックして操作を進めてください。
(画面上のMC8541Pはお使いのボード名に置き換えてご覧ください。)



- ③ デバイスドライバのアンインストール画面
【次へ】のボタンをクリック（画面上のMC8541Pはお使いのボード名に置き換えてご覧ください。)



- ④ 処理が終了すると自動的に画面が閉じられます。（アンインストールに時間がかかることがあります。）



- ⑤ 【OK】 ボタンをクリックします。



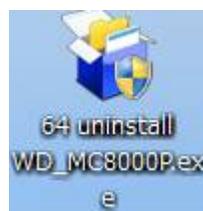
続いて、「2.5.1.2 Uninstall WD_MC8000P.exeの実行」を行います。

(アンインストールしないボードがある場合は、「2.5.1.2 Uninstall WD_MC8000P.exeの実行」を行わず、「2.5.1.3 デバイスマネージャーの確認」を行ってください。)

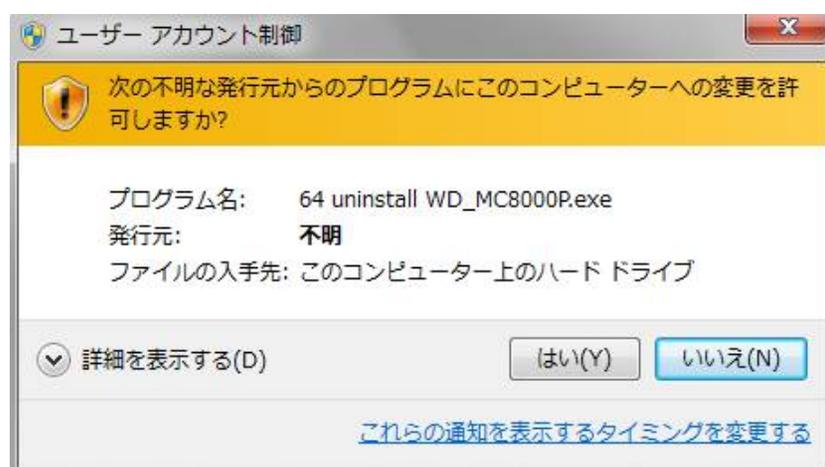
2.5.1.2 Uninstall WD_MC8000P.exeの実行

Uninstall WD_MC8000P.exeを以下の手順で実行します。

- ① 提供CD-ROMのDriverフォルダ（提供CD-ROMがDドライブにある場合は、D:\Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、64 Driverフォルダ内の「64 uninstall WD_MC8000P.exe」をダブルクリックしてください。
(32bitの場合は32 Driverフォルダ内の「32 uninstall WD_MC8000P.exe」をダブルクリックしてください。)



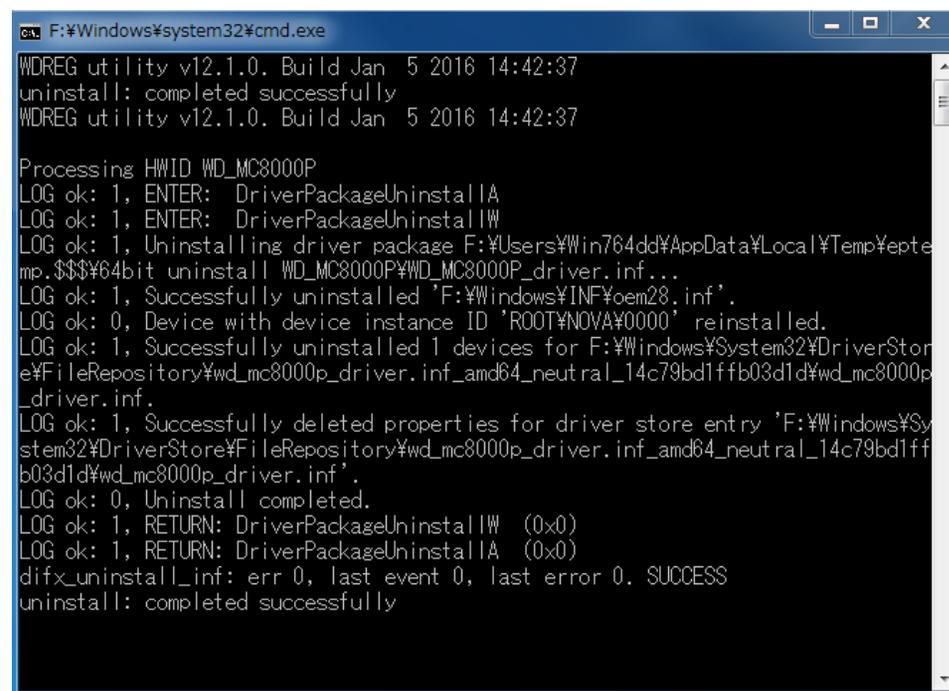
- ② ”ユーザーアカウント制御” の画面が出てきた場合は、【許可】 ボタンをクリックして操作を進めてください。



- ③ デバイスドライバのアンインストール画面
【次へ】のボタンをクリック



- ④ 処理が終了すると自動的に画面が閉じられます。(アンインストールに時間がかかることがあります。完了のメッセージがあるまでお待ちください。)

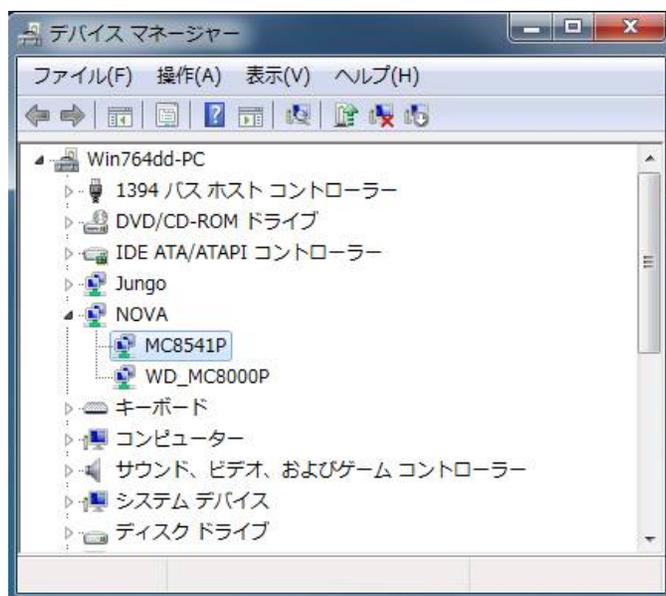


- ⑤ 【OK】ボタンをクリックしてデバイスドライバのアンインストールは完了です。

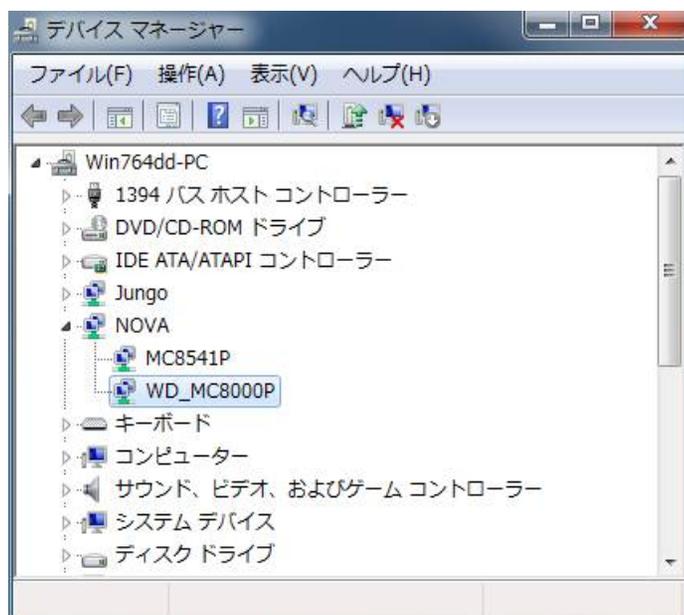


2.5.1.3 デバイスマネージャーの確認

- ① [コントロールパネル]-[システムとセキュリティ]-[システム]-[デバイスマネージャー]タブで本ボードが削除されていることを確認して下さい。
取り外すボードがMC8541Pとし、uninstall MC8541P INF.exeを実行した場合は、図中の反転部分（MC8541P）が削除されていることを確認して下さい。



- ② 「2.5.1.2 Uninstall WD_MC8000P.exeの実行」を行った場合は、図中の反転部分、WD_MC8000Pが削除されていることを確認して下さい。



- ③ 削除されていない場合は、削除されていないドライバをクリックし、[操作]メニューから[削除]を選択後、「このデバイスのドライバソフトウェアを削除する」にチェックしてから、【OK】のボタンをクリックしてドライバを削除して下さい。



- ④ パソコン本体の電源がOFFしてから、外装カバー、スロットカバー等を外します。
⑤ ボードを止めているビスを外します。
⑥ 本ボードを指先でつまんで軽く左右にゆするようにながら引き出し、PCから外します。

2.5.2 異なる種類のボードをインストールしている場合のアンインストール

異なる種類のボードがインストールされている場合（MC8043PとMC8541Pの2種類のボードがインストールされている場合など）のアンインストール方法について説明します。

- ① 「2.5.1.1 Uninstall XXXX INF.exeの実行」を参照して、アンインストールしたいボードの「uninstall XXXX INF.exe」を実行します
- ② 「2.5.1.3 デバイスマネージャーの確認」の②以外の作業を行います。
アンインストールしたいボードの種類が複数ある場合は、①、②の作業を繰り返し行ってください。
- ③ 続いて、全てのボードのアンインストールを行う場合のみ、「2.5.1.2 Uninstall WD_MC8000P.exeの実行」を参照して、「Uninstall WD_MC8000P.exe」を実行します。
- ④ 「2.5.1.3 デバイスマネージャーの確認」を行います。

2.6 デバイスドライバの更新

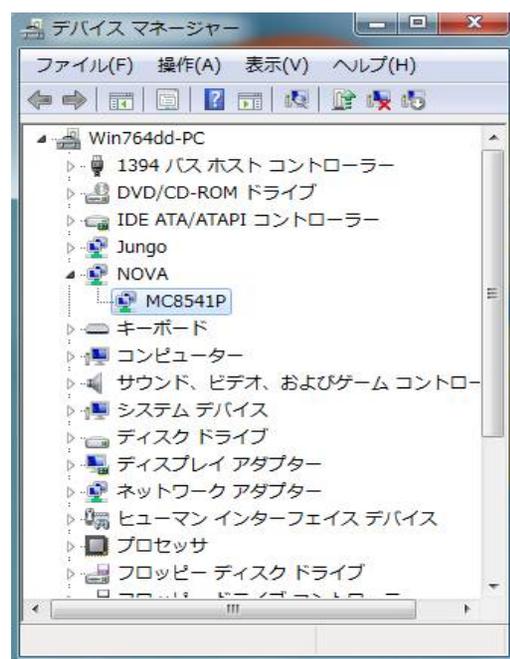
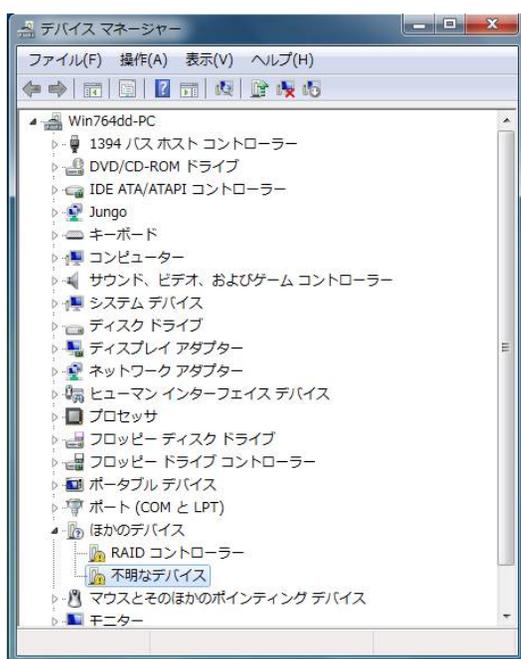
デバイスドライバのバージョンが新しくなった場合、次の手順でドライバの更新を行ってください。
以下に、更新手順を説明します。

起動中の他のアプリケーションは終了させてください。

デバイスドライバのインストールは必ずアドミニストレーター権限をもったユーザーログインで行ってください。アドミニストレーター権限以外でインストールをした場合正常にインストールされません。

下記の手順はWindows 7を例にしていますが、Windows 8.1、Windows 10でもデバイス更新の進め方は同じです。

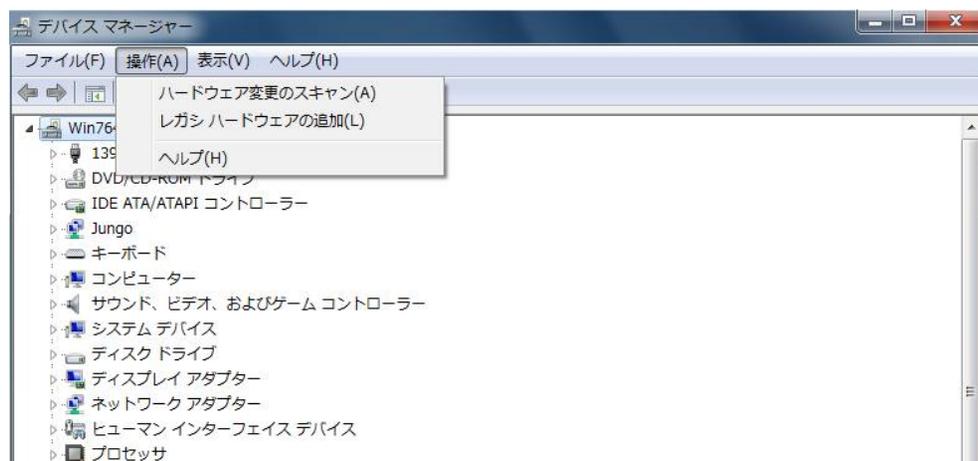
- ① 2.5の方法で現在インストールされているデバイスドライバを削除してください。
- ② パソコンを再起動してください。
- ③ [コントロールパネル]-[システムとセキュリティ]-[システム]-[デバイスマネージャー]タブを開き、①で削除したドライバが削除されているかを確認してください。
削除されている場合は、下記左図の反転部分のように「不明なデバイス」と表示されます。削除されていない場合は下記右図の反転部分のように、ボード名やWD_MC8000Pが表示されます。



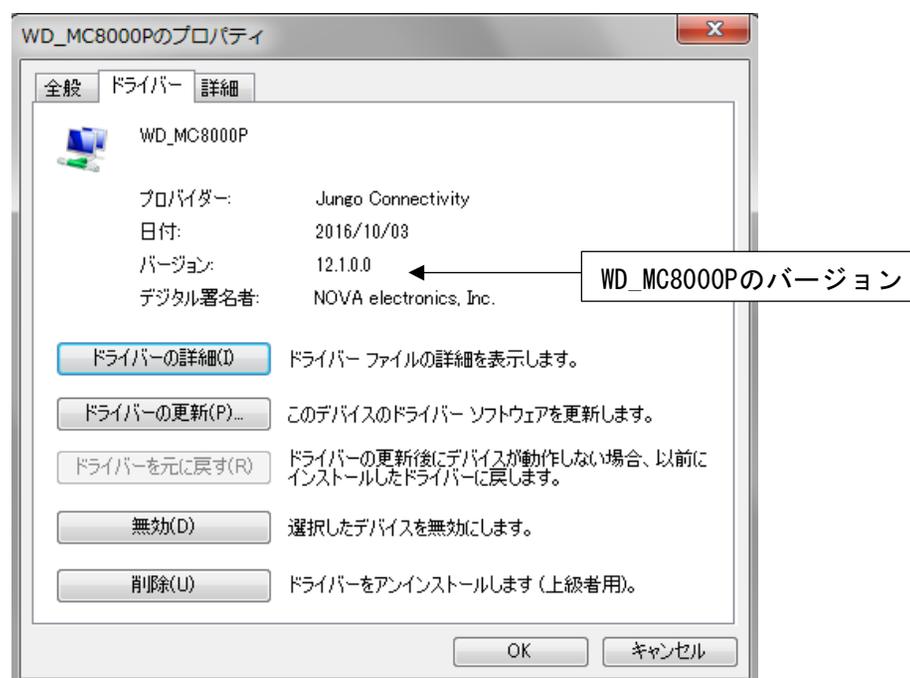
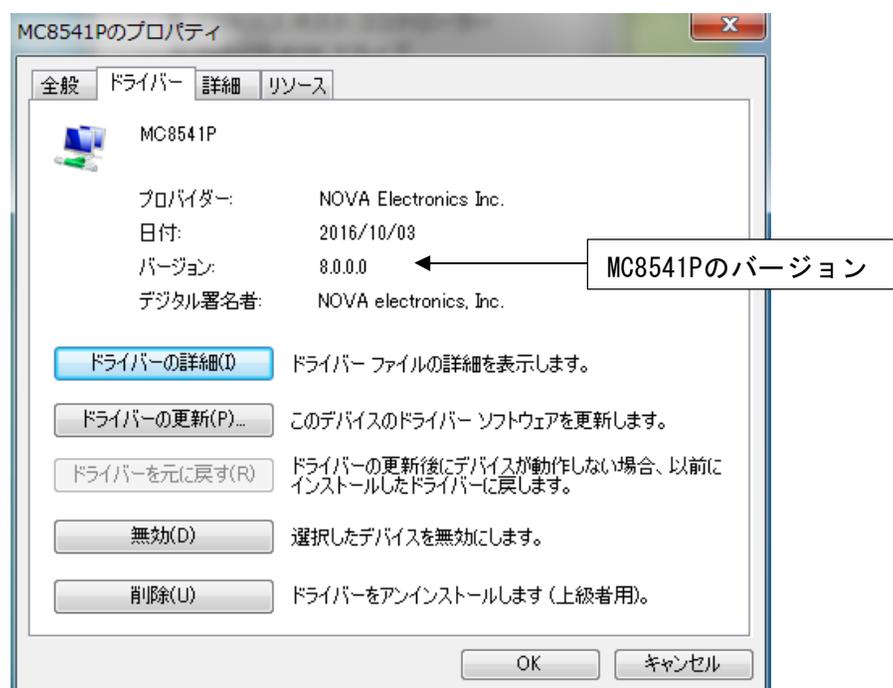
削除されていない場合は、削除されていないドライバをクリックし、[操作]メニューから[削除]を選択後、「このデバイスのドライバソフトウェアを削除する」にチェックしてから、【OK】のボタンをクリックしてドライバを削除します。



- ④ [操作]メニューから[ハードウェア変更のスキャン]を実行して、再び③の作業を行ってください。削除が正しく行なわれるまで、③→④の作業を繰り返してください。



- ⑤ 正しく削除されていることが確認できたら 2.1の方法でインストールするデバイスドライバを準備して下さい。
 ⑥ 2.4の⑤から⑩を参照してインストールします。
 ⑦ 提供CD-ROMのDriverフォルダ（提供CD-ROMがDドライブにある場合は、D:\Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダの¥64 Driver¥Version.txtファイルの「1. ドライババージョン」のバージョンが下記画面のバージョンと一致しているか確認して下さい。
 (32bitの場合は、32 Dirver¥Version.txtファイルで確認して下さい。)
 (画面上のMC8541Pはお使いのボード名に置き換えてご覧下さい。)



3. プログラミング

この章では、アプリケーション開発のためのソフトウェア仕様とプログラミング方法について説明します。
アプリケーション開発は、Microsoft Visual C++ (以下VC++)、Microsoft Visual Basic(以下VB)、あるいはMicrosoft Visual C# (以下C#)のいずれかを使用して行います。

3.1 動作環境

対応OS Windows 7(32bit, 64bit) Windows 8.1(32bit, 64bit) Windows 10(32bit, 64bit)

対応言語

Microsoft Visual C++ 2008 以降
Microsoft Visual Basic 2008 以降
Microsoft Visual C# 2008 以降

注意：

アプリケーション開発を行う場合は、開発ツールのサポート状況などMSDNのサポートページを参考にソフトウェアを開発してください。サンプルプログラムをVisual Studio 2010以降で動作させる場合は、MSDNのサポートページを参考にサンプルプログラムの移行を行ってください。64bitのデバイスドライバは64bitアプリにのみ、32bitのデバイスドライバは32bitアプリにのみ対応しています。

MC8082PeはMC8082Pと、MC8043PeはMC8043Pと同じファイルを使用してください。

MC8541PeはMC8541Pと、MC8581PeはMC8581Pと同じファイルを使用してください。

MC8022P/MC8042PはMC8082Pのファイルを使用してください。

3.2 ソフトウェア構成

3.2.1 ソフトウェア一覧

項目	フォルダ	ファイル名	説明
デバイスドライバ	Driver¥ 32 Driver	32 install MC8043P INF.exe	32bit用MC8043Pデバイスドライバインストールプログラム
		32 install MC8082P INF.exe	32bit用MC8082Pデバイスドライバインストールプログラム
		32 install MC8541P INF.exe	32bit用MC8541Pデバイスドライバインストールプログラム
		32 install MC8581P INF.exe	32bit用MC8581Pデバイスドライバインストールプログラム
		32 uninstall MC8043P INF.exe	32bit用MC8043Pデバイスドライバアンインストールプログラム
		32 uninstall MC8082P INF.exe	32bit用MC8082Pデバイスドライバアンインストールプログラム
		32 uninstall MC8541P INF.exe	32bit用MC8541Pデバイスドライバアンインストールプログラム
		32 uninstall MC8581P INF.exe	32bit用MC8581Pデバイスドライバアンインストールプログラム
		uninstall WD_MC8000P.exe	32bit用WD_MC8000Pアンインストールプログラム

項目	フォルダ	ファイル名	説明
デバイスドライバ	Driver¥ 64 Driver	64 install MC8043P INF.exe	64bit用MC8043Pデバイスドライバインストールプログラム
		64 install MC8082P INF.exe	64bit用MC8082Pデバイスドライバインストールプログラム
		64 install MC8541P INF.exe	64bit用MC8541Pデバイスドライバインストールプログラム
		64 install MC8581P INF.exe	64bit用MC8581Pデバイスドライバインストールプログラム
		64 uninstall MC8043P INF.exe	64bit用MC8043Pデバイスドライバアンインストールプログラム
		64 uninstall MC8082P INF.exe	64bit用MC8082Pデバイスドライバアンインストールプログラム
		64 uninstall MC8541P INF.exe	64bit用MC8541Pデバイスドライバアンインストールプログラム
		64 uninstall MC8581P INF.exe	64bit用MC8581Pデバイスドライバアンインストールプログラム
		64 uninstall WD_MC8000P.exe	64bit用WD_MC8000Pアンインストールプログラム
ライブラリ	Lib¥VC ¥32 lib	MC8000P.LIB	MC8000P.DLLを使用するためのライブラリ 32bit、VC++専用
		MC8000P_DLL.h	MC8000P.DLLを使用するためのヘッダ定義ファイル VC++専用
	Lib¥VC ¥64 lib	MC8000P.LIB	MC8000P.DLLを使用するためのライブラリ 64bit、VC++専用
		MC8000P_DLL.h	MC8000P.DLLを使用するためのヘッダ定義ファイル VC++専用
	Lib¥VB.NET	MC8000P_DLL.vb	MC8000P.DLLを使用するためのDeclare宣言ファイル VB.NET用
Lib¥C#	Mc8000pWrap.dll	MC8000P.DLLを使用するためのクラスライブラリ C#専用	
サンプルプログラム	当社ホームページ (URL: http://www.novaelec.co.jp/) よりダウンロードして下さい。		
MCX304搭載ボード 評価ツール ※MC8082P	Tool ¥MCX304 Board ¥64 Tool (¥32 Tool)	MCX304-A.exe	MCX304搭載ボード評価ツール。画面からパラメータ、モード等を設定し、各コマンドを実行するアプリケーション。
		MCX304-B.exe	
MCX314As搭載ボード 評価ツール ※MC8043P	Tool ¥MCX314As Board ¥64 Tool (¥32 Tool)	ParameterSample	パラメータサンプルファイル： MCX304-x.exeにてこのファイルをロードし、プロットを実行すると、各ファイル名称の動作をプロット画面上で確認できる
		MCX314As-A.exe	MCX314As搭載ボード評価ツール。画面からパラメータ、モード等を設定し、各コマンドを実行するアプリケーション。
MCX514搭載ボード 評価ツール ※MC8541P、MC8581P	Tool ¥MCX514 Board ¥64 Tool (¥32 Tool)	ParameterSample	
		MCX514.exe	MCX514搭載ボード評価ツール。画面からパラメータ、モード等を設定し、各コマンドを実行するアプリケーション。

備考：VC++のMFC AppWizerdが自動的に作成するファイルに関しては説明を省略します。

評価ツールについては、32bitOSでは32bitフォルダ(¥32 Tool)、64bitOSでは64bitフォルダ(¥64 Tool)の実行ファイルをご使用ください。

3.2.2 サンプルプログラムおよび評価ツールの実行時にエラーが発生する場合の対応

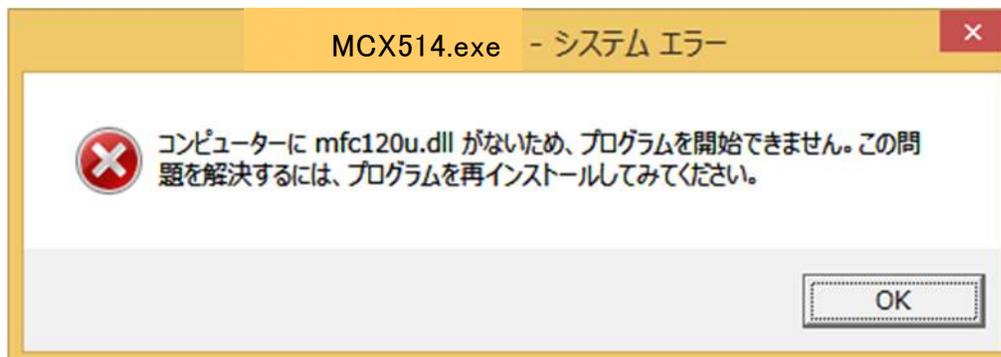
■ VC++で作成されたexe実行時のエラー

本ボードをインストールしたPCにVisual Studioがインストールされていない場合、exeファイル実行時に下図のようなエラーが発生し、実行できない場合があります。エラーが発生した場合、マイクロソフトホームページから「再配布可能パッケージ」をインストールしてください。サンプルプログラムはVisual Studio 2008用、評価ツールはVisual Studio 2013用のものをインストールしてください。また、32bitOS版、64bitOS版がありますので、それぞれに対しPCに対応したものをインストールしてください。例えば、64bitOSで評価ツールのエラーを解除する場合は、Visual Studio 2013の64bit版の「再配布可能パッケージ」をインストールすることになります。また、サンプルプログラムのエラーを解除する場合は、Visual Studio 2008の64bit版の「再配布可能パッケージ」をインストールする必要があります。

注意：エラーの詳細と解除方法の詳細については、必ずマイクロソフトのサポートページを参照し、マイクロソフトサポートの指示に従ってエラーを解除してください。



VC++サンプルプログラム実行時のエラー例



評価ツール実行時のエラー例

■ VB.NETで作成されたexe実行時のエラー

本ボードをインストールしたPCに.NET Framework 3.5がインストールされていない場合、exeファイル実行時に下図のようなエラーが発生し、実行できない場合があります。エラーが発生した場合、画面に従って.NET Framework 3.5 をインストールするか、[コントロールパネル]-[プログラム]-[Windowsの機能と有効化または無効化]で.NET Framework 3.5を有効にしてください。

注意：エラーの詳細と解除方法の詳細については、必ずマイクロソフトのサポートページを参照し、マイクロソフトサポートの指示に従ってエラーを解除してください。



VB.NETサンプルプログラム実行時のエラー例

3.3 開発手順

3.3.1 VC++の場合

アプリケーションはMC8000P.libとMC8000P_DLL.hファイルを使用します。この2ファイルは VC++ 2008 以降対応です。

- ① 32bitOSの場合は¥Lib¥VC¥32 libフォルダ、64bitOSの場合は¥Lib¥VC¥64 libに入っている2つのファイルMC8000P.libとMC8000P_DLL.hを開発するアプリケーションのフォルダにコピーしてください。
- ② VC++の総合開発環境にてMC8000P_DLL.h をご使用のプロジェクトに追加登録してください。
また、API 関数を使用するソースファイルにMC8000P_DLL.h をincludeして下さい。
- ③ [プロジェクト]—[プロパティ]画面で[リンカ]—[入力]を選択し「追加の依存ファイル」にMC8000P.libを追加して下さい。
- ④ 4.1 APIまたは5.1 APIの関数を使用してプログラミングを行って下さい。

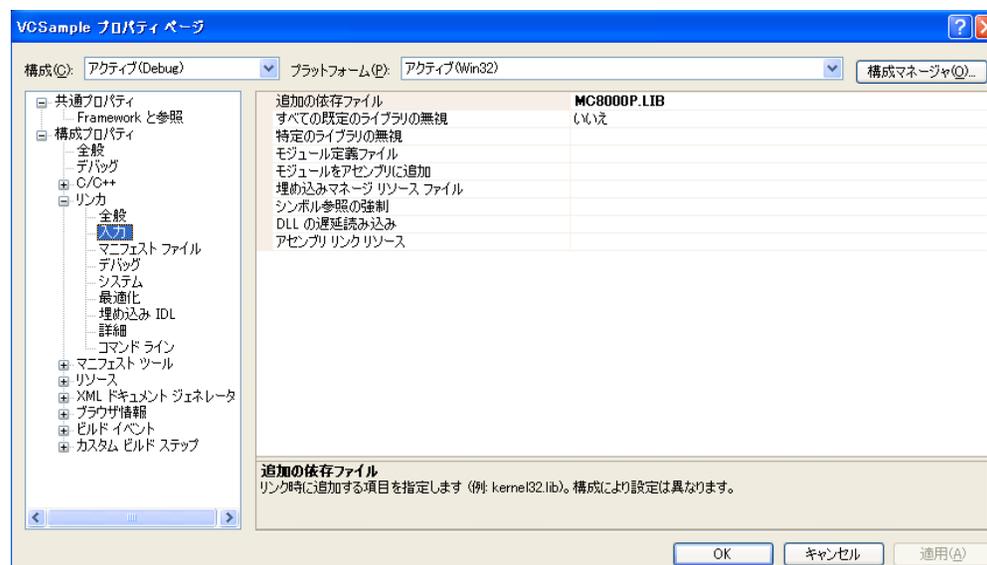


図3.3-1 VC++ 2008 プロジェクトのプロパティ

3.3.2 VB.NETの場合

アプリケーションはMC8000P_DLL.vbファイルを使用します。このファイルは VB2008 以降対応です。

- ① ¥Lib¥VB.NET フォルダに入っているMC8000P_DLL.vbファイルを開発するアプリケーションのプロジェクトに追加してください。
- ② 4.1 APIまたは5.1 APIの関数を使用してプログラミングを行って下さい。

注意：VB.NET では、本ボードに関する割り込みを使用することはできません。

3.3.3 C#の場合

MC8000Pアプリケーションでは、NETに対応したC#クラスライブラリMc8000pWrap.dllを使用します。

- ① ¥LIB¥Csharpフォルダに入っているファイルMc8000pWrap.dllを開発するアプリケーションのフォルダにコピーしてください。
- ② [プロジェクト]—[参照の追加]で「参照」タブを選択しMc8000pWrap.dllへの参照を追加して下さい。
- ③ アプリケーションのソースファイルにusingでネームスペース Mc8000pWrap を追加して下さい。
- ④ 4.1 APIまたは5.1 APIの関数を使用してプログラミングを行って下さい。



図 3.3-2 C# 2008 参照の追加

4. MC8000Pシリーズ ボード

この章では、以下のボードで使用できるAPIについて説明します。

ボード	搭載 I C	I C 数	軸数
MC8043P / MC8043Pe	MCX314As	1	4
MC8082P / MC8082Pe	MCX304	2	8
MC8022P	MCX304	1	2
MC8042P	MCX304	1	4

4.1 API

MC8000P.SYS、MC8000P.DLL がアプリケーションに提供するAPI

4.1.1 関数一覧

下表は、API関数の一覧表です。

「VC」「VB.NET」「C#」の欄は、各言語において各関数を使用できるかどうかを記載しています。

VC++の場合 [VC] の項目を参照して下さい。

VB.NETの場合 [VB.NET] の項目を参照して下さい。

C#の場合 [C#] の項目を参照して下さい。

○は使用できます。×は使用できません。

(1) 基本関数

関数名	説明	VC	VB.NET	C#	ページ	備考
Nmc_Open	ボードの使用を開始する	○	○	○	21	
Nmc_Close	ボードの使用を終了する	○	○	○	〃	
Nmc_CloseAll	全てのボードの使用を終了する	○	○	○	〃	
Nmc_GetBoardInfo	オープンしたボードの情報を取得する	○	○	○	22	
Nmc_OutPort	出力ポートにデータを書く	○	○	○	〃	
Nmc_InPort	入力ポートからデータを読む	○	○	○	23	
Nmc_WriteReg	ボードのレジスタにデータを書き込む	○	○	○	〃	
Nmc_ReadReg	ボードのレジスタからデータを読み込む	○	○	○	24	
Nmc_SetEvent	割り込みを処理するユーザー関数を設定する	○	×	○	25	
Nmc_ResetEvent	割り込みを処理するユーザー関数の設定を解除する	○	×	○	26	
Nmc_ReadEvent	割り込み発生直後の各軸RR3の値を取得する	○	×	○	〃	

(2) リセット、命令

関数名	説明	VC	VB.NET	C#	ページ	備考
Nmc_Reset	ボードに搭載しているICをリセットする	○	○	○	27	
Nmc_Command	指定軸の命令を実行する	○	○	○	〃	
Nmc_Command_IP	補間命令を実行する	○	○	○	〃	※1

(3) ライトレジスタ

関数名	説明	VC	VB.NET	C#	ページ	備考
Nmc_WriteReg0	WR0 (コマンドレジスタ) 書き込み	○	○	○	28	
Nmc_WriteReg1	WR1 (モードレジスタ1) 書き込み	○	○	○	〃	
Nmc_WriteReg2	WR2 (モードレジスタ2) 書き込み	○	○	○	〃	
Nmc_WriteReg3	WR3 (モードレジスタ3) 書き込み	○	○	○	29	
Nmc_WriteReg4	WR4 (アウトプットレジスタ) 書き込み	○	○	○	〃	
Nmc_WriteReg5	WR5 書き込み	○	○	○	〃	
Nmc_WriteReg6	WR6 (ライトデータレジスタ1) 書き込み	○	○	○	30	
Nmc_WriteReg7	WR7 (ライトデータレジスタ2) 書き込み	○	○	○	〃	

(4) リードレジスタ

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_ReadReg0	RR0 (主ステータスレジスタ) 読み出し	○	○	○	30	
Nmc_ReadReg1	RR1 (ステータスレジスタ1) 読み出し	○	○	○	31	
Nmc_ReadReg2	RR2 (ステータスレジスタ2) 読み出し	○	○	○	〃	
Nmc_ReadReg4	RR4 (インプットレジスタ1) 読み出し	○	○	○	〃	
Nmc_ReadReg5	RR5 (インプットレジスタ2) 読み出し	○	○	○	32	
Nmc_ReadReg6	RR6 (リードデータレジスタ1) 読み出し	○	○	○	〃	
Nmc_ReadReg7	RR7 (リードデータレジスタ2) 読み出し	○	○	○	〃	

(5) パラメータ設定

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_Range	レンジ設定	○	○	○	33	
Nmc_Jerk	加速度増加率設定 (加加速度)	○	○	○	〃	
Nmc_Acc	加速度設定	○	○	○	〃	
Nmc_Dec	減速度設定	○	○	○	34	
Nmc_StartSpd	初速度設定	○	○	○	〃	
Nmc_Speed	ドライブ速度設定	○	○	○	〃	
Nmc_Pulse	出力パルス数/補間終点設定 (VC, C#用)	○	×	○	35	
Nmc_Pulse_VB	出力パルス数/補間終点設定 (VB用)	×	○	×	〃	
Nmc_DecP	マニュアル減速点設定 (VC, C#用)	○	×	○	36	
Nmc_DecP_VB	マニュアル減速点設定 (VB用)	×	○	×	〃	
Nmc_Center	円弧中心点設定	○	○	○	〃	※1
Nmc_Lp	論理位置カウンタ設定	○	○	○	37	
Nmc_Ep	実位置カウンタ設定	○	○	○	〃	
Nmc_CompP	COMP+レジスタ設定	○	○	○	〃	
Nmc_CompM	COMP-レジスタ設定	○	○	○	38	
Nmc_AccOfst	加速カウンタオフセット設定	○	○	○	〃	
Nmc_DJerk	減速度増加率設定	○	○	○	〃	※1
Nmc_HomeSpd	原点検出速度設定	○	○	○	39	

(6) その他のモード設定

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_ExpMode	拡張モード設定	○	○	○	39	※1
Nmc_SyncMode	同期動作モード設定	○	○	○	40	※1
Nmc_HomeMode	自動原点出しモード設定	○	○	○	〃	※2

(7) データ読み出し

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_ReadLp	論理位置カウンタ読み出し	○	○	○	41	
Nmc_ReadEp	実位置カウンタ読み出し	○	○	○	〃	
Nmc_ReadSpeed	現在ドライブ速度読み出し	○	○	○	〃	
Nmc_ReadAccDec	現在加/減速度読み出し	○	○	○	42	
Nmc_ReadSyncBuff	同期バッファレジスタ読み出し	○	○	○	〃	※1

(8) 状態取得

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_GetDriveStatus	ドライブ状態取得	○	○	○	43	
Nmc_GetCNextStatus	連続補間次データ書込み可能状態取得	○	○	○	〃	※1
Nmc_GetBpSc	B P補間スタックカウンタ取得	○	○	○	44	※1

(9) 書き込み・読み出し

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_WriteRegSetAxis	軸指定ライトレジスタ書き込み (WR1~3)	○	○	○	44	
Nmc_ReadRegSetAxis	軸指定リードレジスタ読み出し (RR1~2)	○	○	○	45	
Nmc_WriteData	データ書き込み (パラメータ)	○	○	○	〃	
Nmc_WriteData2	データ書き込み (拡張モード、同期動作モード)	○	○	○	46	※1
Nmc_ReadData	データ読み出し	○	○	○	〃	

(10) 補間実行

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_2BPExec	2軸B P補間実行	○	○	○	47	※1
Nmc_3BPExec	3軸B P補間実行	○	○	○	49	※1
Nmc_2BPExec_BG	2軸B P補間実行 (バックグラウンドで実行)	○	○	○	51	※1
Nmc_3BPExec_BG	3軸B P補間実行 (バックグラウンドで実行)	○	○	○	54	※1
Nmc_2CIPExec	2軸連続補間実行	○	○	○	57	※1
Nmc_3CIPExec	3軸連続補間実行	○	○	○	59	※1
Nmc_2CIPExec_BG	2軸連続補間実行 (バックグラウンドで実行)	○	○	○	61	※1
Nmc_3CIPExec_BG	3軸連続補間実行 (バックグラウンドで実行)	○	○	○	64	※1
Nmc_IPStop	補間実行を中断する	○	○	○	67	※1
Nmc_IPGetMsgNo	補間終了時の受信メッセージからポート番号とIC番号取得	○	○	○	〃	※1

※1 : MC8043P/MC8043Pe 専用関数

※2 : MC8082P/MC8082Pe、MC8022P、MC8042P 専用関数

4.1.2 関数仕様

VC++の場合 : VC と[VC]の項目を参照して下さい。
 VB.NETの場合 : VB.NET と[VB.NET]の項目を参照して下さい。
 C#の場合 : C#と[C#]の項目を参照して下さい。

指定のない項目は、各言語共通の内容です。

関数名	機能 及び 内容
Nmc_Open	<p>ボードの使用を開始する。</p> <p>VC BOOL Nmc_Open(int No, BOOL IntrptFlg); VB.NET Function Nmc_Open(ByVal No As Integer, ByVal IntrptFlg As Integer) As Integer C# bool MC8000P.Nmc_Open(int No, bool IntrptFlg);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IntrptFlg 割り込みを使用するかを指定する。 [VC] TRUE : 使用する。FALSE : 使用しない。 [VB.NET] False固定。割り込みを使用しない設定。(VBでは割り込みを使用できません) [C#] true : 使用する。false : 使用しない。</p> <p>戻り値 [VC] オープンに成功するとTRUE、失敗するとFALSE [VB.NET] オープンに成功すると0以外、失敗すると0 [C#] オープンに成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] status = Nmc_Open(0, FALSE); // ボード番号0をオープン、割り込みを使用しない [VB.NET] status = Nmc_Open(0, False) [C#] status = MC8000P.Nmc_Open(0, false);</p>
Nmc_Close	<p>ボードの使用を終了する。</p> <p>VC BOOL Nmc_Close(int No); VB.NET Function Nmc_Close(ByVal No As Integer) As Integer C# bool MC8000P.Nmc_Close(int No);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>戻り値 [VC] クローズに成功するとTRUE、失敗するとFALSE [VB.NET] クローズに成功すると0以外、失敗すると0 [C#] クローズに成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] status = Nmc_Close(0); // ボード番号0をクローズ [VB.NET] status = Nmc_Close(0) [C#] status = MC8000P.Nmc_Close(0);</p>
Nmc_CloseAll	<p>全てのボードの使用を終了する。</p> <p>VC BOOL Nmc_CloseAll(void); VB.NET Function Nmc_CloseAll() As Integer C# bool MC8000P.Nmc_CloseAll();</p> <p>入力パラメータ なし</p> <p>戻り値 [VC] クローズに成功するとTRUE、失敗するとFALSE [VB.NET] クローズに成功すると0以外、失敗すると0 [C#] クローズに成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] status = Nmc_CloseAll(); // 全てのボードをクローズ [VB.NET] status = Nmc_CloseAll() [C#] status = MC8000P.Nmc_CloseAll();</p>

関数名	機能 及び 内容
Nmc_GetBoardInfo	<p>オープンしたボードの情報としてデバイスIDを取得する。</p> <pre> VC BOOL Nmc_GetBoardInfo(int No, USHORT* DeviceID); VB.NET Function Nmc_GetBoardInfo(ByVal No As Integer, ByRef DeviceID As Short) As Integer C# bool MC8000P.Nmc_GetBoardInfo(int No, out ushort DeviceID); </pre> <p>入力パラメータ</p> <p>No ボード番号（ボード上のロータリースイッチの値(0~15)）</p> <p>DeviceID [VC] 取得したボードのデバイスIDを格納する変数のアドレス [VB.NET][C#] 取得したボードのデバイスIDを格納する変数 各ボードのデバイスIDは「4.1.3 補足説明」(1)③参照。 [VC][VB.NET] MC8082P/MC8082Pe, 42P, 22PはID_MC8082P、MC8043P/MC8043PeはID_MC8043P。 [C#] MC8082P/MC8082Pe, 42P, 22PはDev_ID.MC8082P、MC8080PはDev_ID.MC8080P、 MC8043P/MC8043PeはDev_ID.MC8043Pを指定する。</p> <p>戻り値</p> <p>[VC] 取得が成功するとTRUE、失敗するとFALSE [VB.NET] 取得が成功すると0以外、失敗すると0 [C#] 取得が成功するとtrue、失敗するとfalse</p> <p>使用例</p> <pre> [VC] USHORT DeviceID; status = Nmc_GetBoardInfo(No, &DeviceID); // デバイスID取得 if(DeviceID == ID_MC8082P) // MC8082Pの場合 [VB.NET] Dim DeviceID As Short status = Nmc_GetBoardInfo(No, DeviceID) If DeviceID = ID_MC8082P Then [C#] ushort DeviceID; status = Nmc_GetBoardInfo(No, out DeviceID); if(DeviceID == Dev_ID.MC8082P) </pre>
Nmc_OutPort	<p>出力ポートに2バイトデータを書き込む。</p> <pre> VC void Nmc_OutPort(int No, long Adr, long Data); VB.NET Sub Nmc_OutPort(ByVal No As Integer, ByVal Adr As Integer, ByVal Data As Integer) C# void MC8000P.Nmc_OutPort(int No, REG_MCX Adr, int Dat); </pre> <p>入力パラメータ</p> <p>No ボード番号（ボード上のロータリースイッチの値(0~15)）</p> <p>Adr 書き込むアドレス。各ボード取扱説明書に記載しているI/Oアドレス。 詳細は「4.1.3 補足説明」(1)①、(6)参照。</p> <p>Data 書き込むデータ</p> <p>戻り値</p> <p>なし</p> <p>使用例</p> <pre> [VC] Nmc_OutPort(No, 0, 0x8000); // IC-Aのソフトリセット(WR0書き込み) [VB.NET] Call Nmc_OutPort(No, 0, &H8000) [C#] MC8000P.Nmc_OutPort(No, REG_MCX.WR0_A, 0x8000); </pre>

関数名	機能 及び 内容
Nmc_InPort	<p>入力ポートから2バイトデータを読み出す。</p> <pre> VC long Nmc_InPort(int No, long Adr); VB.NET Function Nmc_InPort(ByVal No As Integer, ByVal adr As Integer) As Integer C# int MC8000P.Nmc_InPort(int No, REG_MCX Adr); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) Adr 読み出すアドレス。各ボード取扱説明書に記載している I/Oアドレス。 詳細は「4.1.3 補足説明」(1)①、(6)参照。</p> <p>戻り値 入力ポートから読み込んだデータ</p> <p>使用例 [VC] data = Nmc_InPort(No, 0); // IC-Aのリードレジスタ RR0 の読み出し [VB.NET] data = Nmc_InPort(No, 0) [C#] data = MC8000P.Nmc_InPort(No, REG_MCX.RR0_A);</p> <p>注意 [VC][C#] RR3レジスタのデータ読み出しに関しては、Nmc_ReadEvent関数の説明を参考にして下さい。</p>
Nmc_WriteReg	<p>ボードのライトレジスタ(WR0~WR7)にデータを書き込む。</p> <pre> VC void Nmc_WriteReg(int No, int IcNo, long RegNum, long Dat); VB.NET Sub Nmc_WriteReg(ByVal No As Integer, ByVal IcNo As Integer, ByVal RegNum As Integer, ByVal Dat As Integer) C# void MC8000P.Nmc_WriteReg(int No, int IcNo, int RegNum, int Dat); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 RegNum 書き込むレジスタ(MCX_WR0~MCX_WR7)。詳細は「4.1.3 補足説明」(1)参照。 例) [VC][VB.NET] WR0の場合は MCX_WR0 を、WR1の場合は MCX_WR1 を指定する。 [C#] WR0の場合は REG_MCX.WR0 を、WR1の場合は REG_MCX.WR1 を指定する。 Dat 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg(No, 0, MCX_WR0, 0x8000); // IC-Aのソフトリセット(WR0書き込み) [VB.NET] Call Nmc_WriteReg(No, 0, MCX_WR0, &H8000) [C#] MC8000P.Nmc_WriteReg(No, 0, REG_MCX.WR0, 0x8000);</p>

関数名	機能 及び 内容
Nmc_ReadReg	<p>ボードのリードレジスタ (RR0~RR7) からデータを読み出す。</p> <p>VC long Nmc_ReadReg(int No, int IcNo, long RegNum); VB.NET Function Nmc_ReadReg(ByVal No As Integer, ByVal IcNo As Integer, ByVal RegNum As Integer) As Integer C# void MC8000P.Nmc_ReadReg(int No, int IcNo, int RegNum);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 RegNum 読み出すレジスタ (MCX_RR0~MCX_RR7)。詳細は「4.1.3 補足説明」(1)参照。 例) [VC][VB.NET] RR0の場合は MCX_RR0 を、RR1の場合は MCX_RR1 を指定する。 [C#] RR0の場合は REG_MCX.RR0 を、RR1の場合は REG_MCX.RR1 を指定する。</p> <p>戻り値 リードレジスタから読み出したデータ</p> <p>使用例 [VC] data = Nmc_ReadReg (No, 0, MCX_RR0); // IC-Aのリードレジスタ RR0 の読み出し [VB.NET] data = Nmc_ReadReg (No, 0, MCX_RR0) [C#] data = MC8000P.Nmc_ReadReg (No, 0, REG_MCX.WR0);</p> <p>注意 [VC][C#] RR3レジスタのデータ読み出しに関しては、Nmc_ReadEvent関数の説明を参考にして下さい。</p>

関数名	機能 及び 内容														
Nmc_SetEvent	<p>割り込みを処理するユーザー関数を設定する。 この関数を実行すると、割り込みが発生した時にユーザー関数が呼び出され、指定した引数が1つ渡されます。 このユーザー関数は1つのスレッドとして起動されます。 割り込み処理する関数の設定を解除する場合は Nmc_ResetEvent を実行して下さい。</p> <p>VC BOOL Nmc_SetEvent(int No, LPTHREAD_START_ROUTINE UserThread, LPVOID lpParameter); VB.NET 使用できません C# bool MC8000P.Nmc_SetEvent(int No, UserThread Callback, int param);</p> <p>入力パラメータ</p> <table border="0"> <tr> <td>No</td> <td>ボード番号 (ボード上のロータリースイッチの値(0~15))</td> </tr> <tr> <td>[VC] UserThread</td> <td>ユーザー関数のアドレス</td> </tr> <tr> <td>[VC] lpParameter</td> <td>ユーザー関数スレッドに渡す1つの引数を指定する。 スレッドで使用可能なポインタを設定して下さい。 引数を使用しない場合は、NULL等で良い。 ポインタの場合、ユーザー関数呼び出し時に使用可能なポインタを設定して下さい。</td> </tr> <tr> <td>[C#] Callback</td> <td>ユーザーメソッド (デリゲート型) 詳細は「4.1.4使用方法」を参照。</td> </tr> <tr> <td>[C#] param</td> <td>割り込み発生時にユーザー関数に渡す1つのパラメータ (数値等int限定) を指定します。</td> </tr> </table> <p>戻り値</p> <table border="0"> <tr> <td>[VC]</td> <td>成功するとTRUE、失敗するとFALSE</td> </tr> <tr> <td>[C#]</td> <td>成功するとtrue、失敗するとfalse</td> </tr> </table> <p>使用例</p> <p>[VC] (ボード番号0の場合) <pre>status = Nmc_SetEvent(0, MC_EventFunc0, lpParam); // 関数のアドレスと引数を設定 Nmc_WriteReg1(0, 0, AXIS_ALL, 0x8000); // IC-Aの停止時割り込み発生 (全軸)</pre> (ボード番号1の場合) <pre>status = Nmc_SetEvent(1, MC_EventFunc1, NULL); // 関数のアドレスと引数を設定 Nmc_WriteReg1(1, 0, AXIS_ALL, 0x8000); // IC-Aの停止時割り込み発生 (全軸)</pre> ■ 割り込みユーザー関数例 <pre>DWORD WINAPI MC_EventFunc0(LPVOID lpParam) { long Rr3X, Rr3Y, Rr3Z, Rr3U; Nmc_ReadEvent(0, 0, &Rr3X, &Rr3Y, &Rr3Z, &Rr3U); // ボード0, IC-AのRR3割り込みデータ読み出し return 0; } DWORD WINAPI MC_EventFunc1(LPVOID lpParam) { long Rr3X, Rr3Y, Rr3Z, Rr3U; Nmc_ReadEvent(1, 0, &Rr3X, &Rr3Y, &Rr3Z, &Rr3U); // ボード1, IC-AのRR3割り込みデータ読み出し return 0; }</pre> [C#] <pre>// 割り込みユーザーメソッド isr をデリゲート型変数に代入 MC8000P.callback[0] = new MC8000P.UserThread(isr); // 割り込みユーザーメソッドの設定 MC8000P.Nmc_SetEvent(no, MC8000P.callback[0], param);</pre> </p>	No	ボード番号 (ボード上のロータリースイッチの値(0~15))	[VC] UserThread	ユーザー関数のアドレス	[VC] lpParameter	ユーザー関数スレッドに渡す1つの引数を指定する。 スレッドで使用可能なポインタを設定して下さい。 引数を使用しない場合は、NULL等で良い。 ポインタの場合、ユーザー関数呼び出し時に使用可能なポインタを設定して下さい。	[C#] Callback	ユーザーメソッド (デリゲート型) 詳細は「4.1.4使用方法」を参照。	[C#] param	割り込み発生時にユーザー関数に渡す1つのパラメータ (数値等int限定) を指定します。	[VC]	成功するとTRUE、失敗するとFALSE	[C#]	成功するとtrue、失敗するとfalse
No	ボード番号 (ボード上のロータリースイッチの値(0~15))														
[VC] UserThread	ユーザー関数のアドレス														
[VC] lpParameter	ユーザー関数スレッドに渡す1つの引数を指定する。 スレッドで使用可能なポインタを設定して下さい。 引数を使用しない場合は、NULL等で良い。 ポインタの場合、ユーザー関数呼び出し時に使用可能なポインタを設定して下さい。														
[C#] Callback	ユーザーメソッド (デリゲート型) 詳細は「4.1.4使用方法」を参照。														
[C#] param	割り込み発生時にユーザー関数に渡す1つのパラメータ (数値等int限定) を指定します。														
[VC]	成功するとTRUE、失敗するとFALSE														
[C#]	成功するとtrue、失敗するとfalse														

関数名	機能 及び 内容
Nmc_ResetEvent	<p>割り込みを処理するユーザー関数の設定を解除する。 この関数を実行すると、割り込みが発生してもユーザー関数は呼び出されません。</p> <p>VC BOOL Nmc_ResetEvent(int No); VB. NET 使用できません C# bool MC8000P.Nmc_ResetEvent(int No);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>戻り値 [VC] 成功するとTRUE、失敗するとFALSE [C#] 成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] status = Nmc_ResetEvent(No); [C#] MC8000P.Nmc_WriteReg1(No, AXIS.ALL, 0x0000); // 割り込み禁止 (全軸) status = MC8000P.Nmc_ResetEvent(No);</p>
Nmc_ReadEvent	<p>割り込み発生直後の各軸RR3の値を取得する。(ドライバ内のRR3データは読み出し後クリアされる)</p> <p>VC BOOL Nmc_ReadEvent(int No, int IcNo, long* RrIrX, long* RrIrY, long* RrIrZ, long* RrIrU); VB. NET 使用できません C# bool MC8000P.Nmc_ReadEvent(int No, int IcNo, out int RrIrX, out int RrIrY, out int RrIrZ, out int RrIrU);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 RrIrX X軸のRR3を格納する為のバッファへのポインタ RrIrY Y軸のRR3を格納する為のバッファへのポインタ RrIrZ Z軸のRR3を格納する為のバッファへのポインタ RrIrU U軸のRR3を格納する為のバッファへのポインタ</p> <p>戻り値 [VC] 成功するとTRUE、失敗するとFALSE [C#] 成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] long Rr3X[2], Rr3Y[2], Rr3Z[2], Rr3U[2]; Nmc_ReadEvent(No, 0, &Rr3X[0], &Rr3Y[0], &Rr3Z[0], &Rr3U[0]); //IC-AのRR3データを読み出す Nmc_ReadEvent(No, 1, &Rr3X[1], &Rr3Y[1], &Rr3Z[1], &Rr3U[1]); //IC-BのRR3データを読み出す [C#] int Rr3X, Rr3Y, Rr3Z, Rr3U; // X軸、Y軸、Z軸、U軸 MC8000P.Nmc_ReadEvent(No, IcNo, out Rr3X, out Rr3Y, out Rr3Z, out Rr3U);</p> <p>注意 ボードで割り込みが発生した直後ドライバ内でRR3を読み出してしまふのでRR3はクリアされてしまいます。割り込み発生直後のRR3を確認する場合はこの関数を使用してください。 また、Nmc_SetEvent、Nmc_ResetEvent関数の実行とは関係なく、割り込みが発生するとドライバは必ずRR3データを読み出し保存します。ドライバ内に保存されたRR3データは、Nmc_ReadEvent関数を実行して読み出すとクリアされます。 ドライバ内のRR3データをクリアしたい時は、Nmc_ReadEvent関数を実行して下さい。</p>

関数名	機能 及び 内容
Nmc_Reset	<p>ボードに搭載している I C をリセットする。</p> <pre> VC void Nmc_Reset(int No, int IcNo); VB.NET Sub Nmc_Reset(ByVal No As Integer, ByVal IcNo As Integer) C# void MC8000P.Nmc_Reset(int No, int IcNo); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。詳細は4.1.3(7)参照。</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Reset(1, 0); // ボード番号 1 の I C - A をリセットする [VB.NET] Call Nmc_Reset(1, 0) [C#] MC8000P.Nmc_Reset(1, 0);</p>
Nmc_Command	<p>指定軸の命令を実行する。(WROに指定軸の命令を書く)</p> <pre> VC void Nmc_Command(int No, int IcNo, int axis, int cmd); VB.NET Sub Nmc_Command(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal cmd As Integer) C# void MC8000P.Nmc_Command(int No, int IcNo, AXIS axis, CMD cmd); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 Axis 命令を実行する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 cmd 命令番号。定義ファイル(※1)内のコマンド定義の「ドライブ命令、その他の命令」の中から一つを指定する。+方向定量ドライブの場合はCMD_F_DRV_Pを指定する。 [C#]の場合は、「4.1.3 補足説明」(1)④参照。 ※1 : [VC]MC8000P_DLL.H, [VB.NET]MC8000P_DLL.vb</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Command(No, IcNo, AXIS_X, CMD_F_DRV_P); // X軸の+方向定量ドライブを実行する [VB.NET] Call Nmc_Command(No, IcNo, AXIS_X, CMD_F_DRV_P) [C#] MC8000P.Nmc_Command(No, IcNo, AXIS.X, CMD.CMD_F_DRV_P);</p>
Nmc_Command_IP	<p>補間命令を実行する。(WROに補間命令を書く) ※MCX314As専用</p> <pre> VC void Nmc_Command_IP(int No, int IcNo, int cmd); VB.NET Sub Nmc_Command_IP(ByVal No As Integer, ByVal IcNo As Integer, ByVal cmd As Integer) C# void MC8000P.Nmc_Command_IP(int No, int IcNo, CMD cmd); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 cmd 命令番号。定義ファイル(※1)内のコマンド定義の「補間命令」の中から一つを指定する。 2軸直線補間ドライブの場合はCMD_IP_2STを指定する。 [C#]の場合は、「4.1.3 補足説明」(1)④参照。 ※1 : [VC]MC8000P_DLL.H, [VB.NET]MC8000P_DLL.vb</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg5(No, IcNo, 0x0004); // 補間軸設定 (主軸 : X、第2軸 : Y) Nmc_Command_IP(No, IcNo, CMD_IP_2ST); // 2軸直線補間ドライブを実行する [VB.NET] Call Nmc_WriteReg5(No, IcNo, &H0004) Call Nmc_Command_IP(No, IcNo, CMD_IP_2ST) [C#] MC8000P.Nmc_WriteReg5(No, IcNo, 0x0004); MC8000P.Nmc_Command_IP(No, IcNo, CMD.CMD_IP_2ST);</p>

関数名	機能 及び 内容
Nmc_WriteReg0	<p>WR 0 (コマンドレジスタ) にデータを書き込む。</p> <p>VC void Nmc_WriteReg0(int No, int IcNo, long wdata); VB.NET Sub Nmc_WriteReg0(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg0(int No, int IcNo, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg0(No, IcNo, 0x0120); // X軸の+方向定量ドライブを実行する [VB.NET] Call Nmc_WriteReg0(No, IcNo, &H120) [C#] MC8000P.Nmc_WriteReg0(No, IcNo, 0x0120);</p>
Nmc_WriteReg1	<p>WR 1 (モードレジスタ 1) にデータを書き込む。</p> <p>VC void Nmc_WriteReg1(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_WriteReg1(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg1(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「4.1.3 補足説明」(2)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg1(No, IcNo, AXIS_X, 0x8000); // 停止時割り込み発生(X軸) [VB.NET] Call Nmc_WriteReg1(No, IcNo, AXIS_X, &H8000) [C#] MC8000P.Nmc_WriteReg1(No, IcNo, AXIS.X, 0x8000);</p>
Nmc_WriteReg2	<p>WR 2 (モードレジスタ 2) にデータを書き込む。</p> <p>VC void Nmc_WriteReg2(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_WriteReg2(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg2(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「4.1.3 補足説明」(2)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg2(No, IcNo, AXIS_Y, 0x2000); // ALARM 有効(Y軸) [VB.NET] Call Nmc_WriteReg2(No, IcNo, AXIS_Y, &H2000) [C#] MC8000P.Nmc_WriteReg2(No, IcNo, AXIS.Y, 0x2000);</p>

関数名	機能 及び 内容
Nmc_WriteReg3	<p>WR 3 (モードレジスタ 3) にデータを書き込む。</p> <p>VC void Nmc_WriteReg3(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_WriteReg3(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg3(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「4.1.3 補足説明」(2)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg3(No, IcNo, AXIS_ALL, 0x0001); // 全軸マニュアル減速 [VB.NET] Call Nmc_WriteReg3(No, IcNo, AXIS_ALL, &H1) [C#] MC8000P.Nmc_WriteReg3(No, IcNo, AXIS.ALL, 0x0001);</p>
Nmc_WriteReg4	<p>WR 4 (アウトプットレジスタ) にデータを書き込む。</p> <p>VC void Nmc_WriteReg4(int No, int IcNo, long wdata); VB.NET Sub Nmc_WriteReg4(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg4(int No, int IcNo, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg4(No, IcNo, 0x0001); // X軸汎用出力OUT0 Hiレベル出力 [VB.NET] Call Nmc_WriteReg4(No, IcNo, &H0001) [C#] MC8000P.Nmc_WriteReg4(No, IcNo, 0x0001);</p>
Nmc_WriteReg5	<p>WR 5 にデータを書き込む。</p> <p>VC void Nmc_WriteReg5(int No, int IcNo, long wdata); VB.NET Sub Nmc_WriteReg5(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg5(int No, int IcNo, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 ■MCX304 [VC] Nmc_WriteReg5(No, IcNo, 0x0001); // X軸汎用出力OUT0 有効 [VB.NET] Call Nmc_WriteReg5(No, IcNo, &H1) [C#] MC8000P.Nmc_WriteReg5(No, IcNo, 0x0001); ■MCX314As [VC] Nmc_WriteReg5(No, IcNo, 0x0024); // 補間軸設定(主軸: X、第2軸: Y、第3軸: Z) [VB.NET] Call Nmc_WriteReg5(No, IcNo, &H0024) [C#] MC8000P.Nmc_WriteReg5(No, IcNo, 0x0024);</p>

関数名	機能 及び 内容
Nmc_WriteReg6	<p>WR 6 (ライトデータレジスタ 1) にデータを書き込む。</p> <p>VC void Nmc_WriteReg6(int No, int IcNo, long wdata); VB.NET Sub Nmc_WriteReg6(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg6(int No, int IcNo, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg6(No, IcNo, 0x1234); // ライトデータレジスタ 1 にデータ (1234)H を書く [VB.NET] Call Nmc_WriteReg6(No, IcNo, &H1234) [C#] MC8000P.Nmc_WriteReg6(No, IcNo, 0x1234);</p>
Nmc_WriteReg7	<p>WR 7 (ライトデータレジスタ 2) にデータを書き込む。</p> <p>VC void Nmc_WriteReg7(int No, int IcNo, long wdata); VB.NET Sub Nmc_WriteReg7(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg7(int No, int IcNo, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg7(No, IcNo, 0x5678); // ライトデータレジスタ 2 にデータ (5678)H を書く [VB.NET] Call Nmc_WriteReg7(No, IcNo, &H5678) [C#] MC8000P.Nmc_WriteReg7(No, IcNo, 0x5678);</p>
Nmc_ReadReg0	<p>R R 0 (主ステータスレジスタ) のデータを読み出す。</p> <p>VC long Nmc_ReadReg0(int No, int IcNo); VB.NET Function Nmc_ReadReg0(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadReg0(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>戻り値 R R 0 (主ステータスレジスタ) のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg0(No, IcNo); // R R 0 のデータを読む [VB.NET] Data = Nmc_ReadReg0(No, IcNo) [C#] Data = MC8000P.Nmc_ReadReg0(No, IcNo);</p>

関数名	機能 及び 内容
Nmc_ReadReg1	<p>R R 1 (ステータスレジスタ 1) のデータを読み出す。</p> <pre> VC long Nmc_ReadReg1(int No, int IcNo, int axis); VB.NET Function Nmc_ReadReg1(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadReg1(int No, int IcNo, AXIS axis); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「4.1.3 補足説明」(2)参照。</p> <p>戻り値 R R 1 (ステータスレジスタ 1) のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg1(No, IcNo, AXIS_X); // X軸のR R 1のデータを読む [VB.NET] Data = Nmc_ReadReg1(No, IcNo, AXIS_X) [C#] Data = MC8000P.Nmc_ReadReg1(No, IcNo, AXIS.X);</p>
Nmc_ReadReg2	<p>R R 2 (ステータスレジスタ 2) のデータを読み出す。</p> <pre> VC long Nmc_ReadReg2(int No, int IcNo, int axis); VB.NET Function Nmc_ReadReg2(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadReg2(int No, int IcNo, AXIS axis); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「4.1.3 補足説明」(2)参照。</p> <p>戻り値 R R 2 (ステータスレジスタ 2) のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg2(No, IcNo, AXIS_Y); // Y軸のR R 2のデータを読む [VB.NET] Data = Nmc_ReadReg2(No, IcNo, AXIS_Y) [C#] Data = MC8000P.Nmc_ReadReg2(No, IcNo, AXIS.Y);</p>
Nmc_ReadReg4	<p>R R 4 (インプットレジスタ 1) のデータを読み出す。</p> <pre> VC long Nmc_ReadReg4(int No, int IcNo); VB.NET Function Nmc_ReadReg4(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadReg4(int No, int IcNo); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>戻り値 R R 4 (インプットレジスタ 1) のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg4(No, IcNo); // R R 4のデータを読む [VB.NET] Data = Nmc_ReadReg4(No, IcNo) [C#] Data = MC8000P.Nmc_ReadReg4(No, IcNo);</p>

関数名	機能 及び 内容
Nmc_ReadReg5	<p>R R 5 (インプットレジスタ 2) のデータを読み出す。</p> <p>VC long Nmc_ReadReg5(int No, int IcNo); VB.NET Function Nmc_ReadReg5(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadReg5(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>戻り値 R R 5 (インプットレジスタ 2) のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg5 (No, IcNo); // R R 5 のデータを読む [VB.NET] Data = Nmc_ReadReg5 (No, IcNo) [C#] Data = MC8000P.Nmc_ReadReg5 (No, IcNo);</p>
Nmc_ReadReg6	<p>R R 6 (リードデータレジスタ 1) のデータを読み出す。</p> <p>VC long Nmc_ReadReg6(int No, int IcNo); VB.NET Function Nmc_ReadReg6(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadReg6(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>戻り値 R R 6 (リードデータレジスタ 1) のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg6 (No, IcNo); // R R 6 のデータを読む [VB.NET] Data = Nmc_ReadReg6 (No, IcNo) [C#] Data = MC8000P.Nmc_ReadReg6 (No, IcNo);</p>
Nmc_ReadReg7	<p>R R 7 (リードデータレジスタ 2) のデータを読み出す。</p> <p>VC long Nmc_ReadReg7(int No, int IcNo); VB.NET Function Nmc_ReadReg7(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadReg7(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>戻り値 R R 7 (リードデータレジスタ 2) のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg7 (No, IcNo); // R R 7 のデータを読む [VB.NET] Data = Nmc_ReadReg7 (No, IcNo) [C#] Data = MC8000P.Nmc_ReadReg7 (No, IcNo);</p>

関数名	機能 及び 内容
Nmc_Range	<p>レンジを設定する。</p> <pre> VC void Nmc_Range(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Range(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Range(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15))</p> <p>IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>axis データを設定する軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「4.1.3 補足説明」(2)参照。</p> <p>wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例</p> <pre> [VC] Nmc_Range(No, IcNo, AXIS_ALL, 800000); // レンジに 800000 (倍率10) を設定する (全軸) [VB.NET] Call Nmc_Range(No, IcNo, AXIS_ALL, 800000) [C#] MC8000P.Nmc_Range(No, IcNo, AXIS.ALL, 800000); </pre>
Nmc_Jerk	<p>加速度増加率 (加加速度) を設定する。</p> <pre> VC void Nmc_Jerk(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Jerk(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Jerk(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15))</p> <p>IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>axis データを設定する軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「4.1.3 補足説明」(2)参照。</p> <p>wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例</p> <pre> [VC] Nmc_Jerk(No, IcNo, AXIS_X, 1000); // 加速度増加率に 1000 を設定する (X軸) [VB.NET] Call Nmc_Jerk(No, IcNo, AXIS_X, 1000) [C#] MC8000P.Nmc_Jerk(No, IcNo, AXIS.X, 1000); </pre>
Nmc_Acc	<p>加速度を設定する。</p> <pre> VC void Nmc_Acc(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Acc(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Acc(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15))</p> <p>IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>axis データを設定する軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「4.1.3 補足説明」(2)参照。</p> <p>wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例</p> <pre> [VC] Nmc_Acc(No, IcNo, AXIS_Y, 100); // 加速度に 100 を設定する (Y軸) [VB.NET] Call Nmc_Acc(No, IcNo, AXIS_Y, 100) [C#] MC8000P.Nmc_Acc(No, IcNo, AXIS.Y, 100); </pre>

関数名	機能 及び 内容
Nmc_Dec	<p>減速度を設定する。</p> <pre> VC void Nmc_Dec(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Dec(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Dec(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Dec(No, IcNo, AXIS_Z, 100); // 減速度に 100 を設定する(Z軸) [VB.NET] Call Nmc_Dec(No, IcNo, AXIS_Z, 100) [C#] MC8000P.Nmc_Dec(No, IcNo, AXIS.Z, 100);</p>
Nmc_StartSpd	<p>初速度を設定する。</p> <pre> VC void Nmc_StartSpd(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_StartSpd(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_StartSpd(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_StartSpd(No, IcNo, AXIS_U, 100); // 初速度に 100 を設定する(U軸) [VB.NET] Call Nmc_StartSpd(No, IcNo, AXIS_U, 100) [C#] MC8000P.Nmc_StartSpd(No, IcNo, AXIS.U, 100);</p>
Nmc_Speed	<p>ドライブ速度を設定する。</p> <pre> VC void Nmc_Speed(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Speed(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Speed(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Speed(No, IcNo, AXIS_X AXIS_Y, 1000); // ドライブ速度に 1000 を設定する(X, Y軸) [VB.NET] Call Nmc_Speed(No, IcNo, AXIS_X Or AXIS_Y, 1000) [C#] MC8000P.Nmc_Speed(No, IcNo, AXIS.X AXIS.Y, 1000);</p>

関数名	機能 及び 内容
Nmc_DecP	<p>マニュアル減速点を設定する。(VC, C#専用)</p> <p>VC void Nmc_DecP(int No, int IcNo, int axis, ULONG wdata); VB.NET 使用できません C# void MC8000P.Nmc_DecP(int No, int IcNo, AXIS axis, uint wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_DecP(No, IcNo, AXIS_U, 30000); // マニュアル減速点に 30000 を設定する(U軸) [C#] MC8000P.Nmc_DecP(No, IcNo, AXIS.U, 30000);</p>
Nmc_DecP_VB	<p>マニュアル減速点を設定する。(VB.NET専用)</p> <p>VC 使用できません VB.NET Sub Nmc_DecP_VB(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Double) C# 使用できません</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VB.NET] Call Nmc_DecP_VB(No, IcNo, AXIS_X, 40000) ' マニュアル減速点に 40000 を設定する(X軸)</p>
Nmc_Center	<p>円弧中心点を設定する。 ※MC8043P/MC8043Pe専用</p> <p>VC void Nmc_Center(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Center(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Center(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Center(No, IcNo, AXIS_Y, 1500); // 円弧中心点に 1500 を設定する(Y軸) [VB.NET] Call Nmc_Center(No, IcNo, AXIS_Y, 1500) [C#] MC8000P.Nmc_Center(No, IcNo, AXIS.Y, 1500);</p>

関数名	機能 及び 内容
Nmc_Lp	<p>論理位置カウンタを設定する。</p> <p>VC void Nmc_Lp(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Lp(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Lp(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Lp(No, IcNo, AXIS_ALL, 0); // 全軸の論理位置カウンタを0クリアする [VB.NET] Call Nmc_Lp(No, IcNo, AXIS_ALL, 0) [C#] MC8000P.Nmc_Lp(No, IcNo, AXIS.ALL, 0);</p>
Nmc_Ep	<p>実位置カウンタを設定する。</p> <p>VC void Nmc_Ep(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Ep(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Ep(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Ep(No, IcNo, AXIS_ALL, 0); // 全軸の実位置カウンタを0クリアする [VB.NET] Call Nmc_Ep(No, IcNo, AXIS_ALL, 0) [C#] MC8000P.Nmc_Ep(No, IcNo, AXIS.ALL, 0);</p>
Nmc_CompP	<p>COMP+レジスタを設定する。</p> <p>VC void Nmc_CompP(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_CompP(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_CompP(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_CompP(No, IcNo, AXIS_X, 50000); // COMP+レジスタに 50000 を設定する(X軸) [VB.NET] Call Nmc_CompP(No, IcNo, AXIS_X, 50000) [C#] MC8000P.Nmc_CompP(No, IcNo, AXIS.X, 50000);</p>

関数名	機能 及び 内容
Nmc_CompM	<p>COMPレジスタを設定する。</p> <pre> VC void Nmc_CompM(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_CompM(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_CompM(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_CompM(No, IcNo, AXIS_X, -50000); // COMPレジスタに -50000 を設定する(X軸) [VB.NET] Call Nmc_CompM(No, IcNo, AXIS_X, -50000) [C#] MC8000P.Nmc_CompM(No, IcNo, AXIS.X, -50000);</p>
Nmc_AccOfst	<p>加速カウンタオフセットを設定する。</p> <pre> VC void Nmc_AccOfst(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_AccOfst(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_AccOfst(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_AccOfst(No, IcNo, AXIS_Y, 20); // 加速カウンタオフセットに 20 を設定する(Y軸) [VB.NET] Call Nmc_AccOfst(No, IcNo, AXIS_Y, 20) [C#] MC8000P.Nmc_AccOfst(No, IcNo, AXIS.Y, 20);</p>
Nmc_DJerk	<p>減速度増加率を設定する。 ※MC8043P/MC8043Pe専用</p> <pre> VC void Nmc_DJerk(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_DJerk(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_DJerk(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_DJerk(No, IcNo, AXIS_Z, 1000); // 減速度増加率に 1000 を設定する(Z軸) [VB.NET] Call Nmc_DJerk(No, IcNo, AXIS_Z, 1000) [C#] MC8000P.Nmc_DJerk(No, IcNo, AXIS.Z, 1000);</p>

関数名	機能 及び 内容
Nmc_HomeSpd	<p>原点検出速度を設定する。</p> <pre> VC void Nmc_HomeSpd(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_HomeSpd(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_HomeSpd(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。</p> <p>wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例</p> <pre> [VC] Nmc_HomeSpd(No, IcNo, AXIS_U, 200); // 原点検出速度に 200 を設定する(U軸) [VB.NET] Call Nmc_HomeSpd(No, IcNo, AXIS_U, 200) [C#] MC8000P.Nmc_HomeSpd(No, IcNo, AXIS.U, 200); </pre>
Nmc_ExpMode	<p>拡張モードを設定する。 ※MC8043P/MC8043Pe専用</p> <pre> VC void Nmc_ExpMode(int No, int IcNo, int axis, long EM6_data, long EM7_data); VB.NET Sub Nmc_ExpMode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal EM6_data As Integer, ByVal EM7_data As Integer) C# void MC8000P.Nmc_ExpMode(int No, int IcNo, AXIS axis, int EM6_data, int EM7_data); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照</p> <p>axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。</p> <p>EM6_data 拡張モードレジスタ EM6 に設定するデータ</p> <p>EM7_data 拡張モードレジスタ EM7 に設定するデータ</p> <p>戻り値 なし</p> <p>使用例 X軸拡張モードに、全フィルタ有効, 遅延512μs, 自動原点出しステップ1, 2, 4の実行を設定する</p> <pre> [VC] Nmc_ExpMode(No, IcNo, AXIS_X, 0x5F00, 0x0045); [VB.NET] Call Nmc_ExpMode(No, IcNo, AXIS_X, &H5F00, &H0045) [C#] MC8000P.Nmc_ExpMode(No, IcNo, AXIS.X, 0x5F00, 0x0045); </pre>

関数名	機能 及び 内容
Nmc_SyncMode	<p>同期動作モードを設定する。 ※MC8043P/MC8043Pe専用</p> <p>VC void Nmc_SyncMode(int No, int IcNo, int axis, long SM6_data, long SM7_data); VB.NET Sub Nmc_SyncMode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal SM6_data As Integer, ByVal SM7_data As Integer) C# void MC8000P.Nmc_SyncMode(int No, int IcNo, AXIS axis, int SM6_data, int SM7_data);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 SM6_data 同期動作モードレジスタ SM6 に設定するデータ SM7_data 同期動作モードレジスタ SM7 に設定するデータ</p> <p>戻り値 なし</p> <p>使用例 「X軸ドライブ終了時にY軸+方向定量ドライブを開始する」を設定する。 ① X軸同期動作モードに、起動要因 D-END でY軸起動を設定する ② Y軸同期動作モードに、動作として+方向定量ドライブ(FDRV+)を設定する</p> <p>[VC] ① Nmc_SyncMode(No, IcNo, AXIS_X, 0x2020, 0); ② Nmc_SyncMode(No, IcNo, AXIS_Y, 0, 0x0001); [VB.NET] ① Call Nmc_SyncMode(No, IcNo, AXIS_X, &H2020, 0) ② Call Nmc_SyncMode(No, IcNo, AXIS_Y, 0, &H0001) [C#] ① MC8000P.Nmc_SyncMode(No, IcNo, AXIS.X, 0x2020, 0); ② MC8000P.Nmc_SyncMode(No, IcNo, AXIS.Y, 0, 0x0001);</p>
Nmc_HomeMode	<p>自動原点出しモードを設定する。 ※MC8082P/MC8082Pe、MC8022P、MC8042P専用関数</p> <p>VC void Nmc_HomeMode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_HomeMode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_HomeMode(int No, int IcNo, AXIS axis, int WR6_data);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照 axis データを設定する軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 WR6_data WR6 に設定するデータ</p> <p>戻り値 なし</p> <p>使用例 X軸自動原点出しモードに、自動原点出しステップ1,2,4の実行を設定する [VC] Nmc_HomeMode(No, IcNo, AXIS_X, 0x0045); [VB.NET] Call Nmc_HomeMode(No, IcNo, AXIS_X, &H0045) [C#] MC8000P.Nmc_HomeMode(No, IcNo, AXIS.X, 0x0045);</p>

関数名	機能 及び 内容
Nmc_ReadLp	<p>論理位置カウンタを読み出す。</p> <pre> VC long Nmc_ReadLp(int No, int IcNo, int axis); VB.NET Function Nmc_ReadLp(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadLp(int No, int IcNo, AXIS axis); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。</p> <p>戻り値 現在の論理位置カウンタの値</p> <p>使用例 [VC] Data = Nmc_ReadLp(No, IcNo, AXIS_X); // X軸の論理位置カウンタを読み出す [VB.NET] Data = Nmc_ReadLp(No, IcNo, AXIS_X) [C#] Data = MC8000P.Nmc_ReadLp(No, IcNo, AXIS.X);</p>
Nmc_ReadEp	<p>実位置カウンタを読み出す。</p> <pre> VC long Nmc_ReadEp(int No, int IcNo, int axis); VB.NET Function Nmc_ReadEp(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadEp(int No, int IcNo, AXIS axis); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。</p> <p>戻り値 現在の実位置カウンタの値</p> <p>使用例 [VC] Data = Nmc_ReadEp(No, IcNo, AXIS_Y); // Y軸の実位置カウンタを読み出す [VB.NET] Data = Nmc_ReadEp(No, IcNo, AXIS_Y) [C#] Data = MC8000P.Nmc_ReadEp(No, IcNo, AXIS.Y)</p>
Nmc_ReadSpeed	<p>現在ドライブ速度を読み出す。</p> <pre> VC long Nmc_ReadSpeed(int No, int IcNo, int axis); VB.NET Function Nmc_ReadSpeed(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadSpeed(int No, int IcNo, AXIS axis); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。</p> <p>戻り値 現在ドライブ速度</p> <p>使用例 [VC] Data = Nmc_ReadSpeed(No, IcNo, AXIS_Z); // Z軸の現在ドライブ速度を読み出す [VB.NET] Data = Nmc_ReadSpeed(No, IcNo, AXIS_Z) [C#] Data = MC8000P.Nmc_ReadSpeed(No, IcNo, AXIS.Z);</p>

関数名	機能 及び 内容
Nmc_ReadAccDec	<p>現在加／減速度を読み出す。</p> <p>ドライブ中の現在加速度、または減速度の値を読み出す。 ドライブ停止時の読み出しデータは不定です。</p> <p>VC long Nmc_ReadAccDec(int No, int IcNo, int axis); VB.NET Function Nmc_ReadAccDec(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadAccDec(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「4.1.3 補足説明」(2)参照。</p> <p>戻り値 現在加／減速度</p> <p>使用例 [VC] Data = Nmc_ReadAccDec(No, IcNo, AXIS_U); // U軸の現在加／減速度を読み出す [VB.NET] Data = Nmc_ReadAccDec(No, IcNo, AXIS_U) [C#] Data = MC8000P.Nmc_ReadAccDec(No, IcNo, AXIS.U);</p>
Nmc_ReadSyncBuff	<p>同期バッファレジスタを読み出す。 ※MC8043P/MC8043Pe専用</p> <p>VC long Nmc_ReadSyncBuff(int No, int IcNo, int axis); VB.NET Function Nmc_ReadSyncBuff(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadSyncBuff(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「4.1.3 補足説明」(2)参照。</p> <p>戻り値 同期バッファレジスタの値</p> <p>使用例 [VC] Data = Nmc_ReadSyncBuff(No, IcNo, AXIS_X); // X軸の同期バッファレジスタを読み出す [VB.NET] Data = Nmc_ReadSyncBuff(No, IcNo, AXIS_X) [C#] Data = MC8000P.Nmc_ReadSyncBuff(No, IcNo, AXIS.X);</p>

関数名	機能 及び 内容
Nmc_GetDriveStatus	<p>ドライブ状態を取得する。指定軸のドライブが終了したか調べる時に使用する。</p> <pre> VC int Nmc_GetDriveStatus(int No, int IcNo, int axis); VB.NET Function Nmc_GetDriveStatus(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_GetDriveStatus(int No, int IcNo, AXIS axis); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis ドライブ状態を取得する軸。複数軸指定可能。 詳細は「4.1.3 補足説明」(2)参照。</p> <p>戻り値 指定した全ての軸のドライブが終了している場合は0を返す。 指定した軸のうち1つ以上がドライブ中の場合は、0以外を返す。</p> <p>使用例</p> <pre> [VC] if(Nmc_GetDriveStatus(No, IcNo, AXIS_X) == 0) // X軸がドライブ終了の場合 AfxMessageBox("X軸ドライブ終了"); else AfxMessageBox("X軸ドライブ中"); [VB.NET] If Nmc_GetDriveStatus(No, IcNo, AXIS_X) = 0 Then Call MsgBox("X軸ドライブ終了") Else Call MsgBox("X軸ドライブ中") End If [C#] if(MC8000P.Nmc_GetDriveStatus(No, IcNo, AXIS.X) == 0) MessageBox.Show("X軸ドライブ終了"); else MessageBox.Show("X軸ドライブ中"); </pre>
Nmc_GetCNextStatus	<p>連続補間次データ書き込み可能状態を取得する。 ※MC8043P/MC8043Pe専用 連続補間実行中に次データ書き込み可能になったか調べる時に使用する。</p> <pre> VC int Nmc_GetCNextStatus(int No, int IcNo); VB.NET Function Nmc_GetCNextStatus(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_GetCNextStatus(int No, int IcNo); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>戻り値 連続補間次データ書き込み可能の場合は、0以外を返す。 連続補間次データ書き込み可能ではない場合は、0を返す。</p> <p>使用例</p> <pre> [VC] If(Nmc_GetCNextStatus(No, IcNo) != 0) // 次データ書き込み可能の場合 AfxMessageBox("連続補間次データ書き込み可能である"); else AfxMessageBox("連続補間次データ書き込み可能ではない"); [VB.NET] If Nmc_GetCNextStatus(No, IcNo) <> 0 Then Call MsgBox("連続補間次データ書き込み可能である") Else Call MsgBox("連続補間次データ書き込み可能ではない") End If [C#] if(MC8000P.Nmc_GetCNextStatus(No, IcNo) != 0) MessageBox.Show("連続補間次データ書き込み可能である"); else MessageBox.Show("連続補間次データ書き込み可能ではない"); </pre>

関数名	機能 及び 内容
Nmc_GetBpSc	<p>B P 補間スタックカウンタの値を取得する。 ※MC8043P/MC8043Pe専用</p> <p>VC int Nmc_GetBpSc(int No, int IcNo);</p> <p>VB.NET Function Nmc_GetBpSc(ByVal No As Integer, ByVal IcNo As Integer) As Integer</p> <p>C# int MC8000P.Nmc_GetBpSc(int No, int IcNo);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>戻り値</p> <p>現在のビットパターン補間スタックカウンタの値</p> <p>使用例</p> <p>[VC] Data = Nmc_GetBpSc(No, IcNo); // B P 補間スタックカウンタの値を取得</p> <p>[VB.NET] Data = Nmc_GetBpSc(No, IcNo)</p> <p>[C#] Data = MC8000P.Nmc_GetBpSc(No, IcNo);</p>
Nmc_WriteRegSetAxis	<p>指定軸の指定したライトレジスタ (WR 1 ~ WR 3 のいずれか) にデータを書き込む。</p> <p>VC void Nmc_WriteRegSetAxis(int No, int IcNo, int axis, int RegNumber, long wdata);</p> <p>VB.NET Sub Nmc_WriteRegSetAxis(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal RegNumber As Integer, ByVal wdata As Integer)</p> <p>C# void MC8000P.Nmc_WriteRegSetAxis(int No, int IcNo, AXIS axis, int RegNumber, int wdata);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照</p> <p>axis データを書き込む軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。</p> <p>RegNumber データを書き込むライトレジスタの番号</p> <p>[VC][VB.NET] RR1はMCX_RR1, RR2はMCX_RR2 を指定する。</p> <p>[C#] RR1はREG_MCX.RR1, RR2はREG_MCX.RR2 を指定する。 詳細は「4.1.3 補足説明」(1)参照</p> <p>wdata 書き込むデータ</p> <p>戻り値</p> <p>なし</p> <p>使用例 全軸のWR 2 にALARM有効(2000)Hを書き込む</p> <p>[VC] Nmc_WriteRegSetAxis(No, IcNo, AXIS_ALL, MCX_WR2, 0x2000);</p> <p>[VB.NET] Call Nmc_WriteRegSetAxis(No, IcNo, AXIS_ALL, MCX_WR2, &H2000)</p> <p>[C#] MC8000P.Nmc_WriteRegSetAxis(No, IcNo, AXIS.ALL, REG_MCX.WR2, 0x2000);</p>

関数名	機能 及び 内容
Nmc_ReadRegSetAxis	<p>指定軸の指定したリードレジスタ (RR 1、RR 2 のどちらか) のデータを読み出す。</p> <p>VC long Nmc_ReadRegSetAxis(int No, int IcNo, int axis, int RegNumber);</p> <p>VB.NET Function Nmc_ReadRegSetAxis(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal RegNumber As Integer) As Integer</p> <p>C# int MC8000P.Nmc_ReadRegSetAxis(int No, int IcNo, AXIS axis, int RegNumber);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15))</p> <p>IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照</p> <p>axis データを読み出す軸。 X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「4.1.3 補足説明」(2)参照。</p> <p>RegNumber データを読み出すリードレジスタの番号 [VC][VB.NET] RR1はMCX_RR1, RR2はMCX_RR2 を指定する。 [C#] RR1はREG_MCX.RR1, RR2はREG_MCX.RR2 を指定する。 詳細は「4.1.3 補足説明」(1)参照</p> <p>戻り値 指定軸の指定したリードレジスタのデータ</p> <p>使用例 X軸のRR1のデータを読み出す [VC] Data = Nmc_ReadRegSetAxis(No, IcNo, AXIS_X, MCX_RR1); [VB.NET] Data = Nmc_ReadRegSetAxis(No, IcNo, AXIS_X, MCX_RR1) [C#] Data = MC8000P.Nmc_ReadRegSetAxis(No, IcNo, AXIS.X, REG_MCX.RR1);</p>
Nmc_WriteData	<p>指定軸に指定したパラメータのデータを書き込む。(データ書き込み命令を実行する)</p> <p>VC void Nmc_WriteData(int No, int IcNo, int axis, int cmd, long wdata);</p> <p>VB.NET Sub Nmc_WriteData(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal cmd As Integer, ByVal wdata As Integer)</p> <p>C# void MC8000P.Nmc_WriteData(int No, int IcNo, AXIS axis, CMD cmd, int wdata);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15))</p> <p>IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>axis データを書き込む軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。</p> <p>cmd データ書き込み命令コード ((00)H~(0E)H, (61)H)。例: レンジ設定は(00)H。 ※MC8082P/MC8082Pe, MC8022P, MC8042PIは(08)H, (0E)Hを除く。 詳細は「4.1.3 補足説明」(1)参照。</p> <p>wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 全軸のドライブ速度に1000を設定する。ドライブ速度の命令コードは(05)H [VC] Nmc_WriteData(No, IcNo, AXIS_ALL, 0x05, 1000); [VB.NET] Call Nmc_WriteData(No, IcNo, AXIS_ALL, &H05, 1000) [C#] MC8000P.Nmc_WriteData(No, IcNo, AXIS.ALL, 0x05, 1000);</p>

関数名	機能 及び 内容
Nmc_WriteData2	<p>指定軸に拡張モード、あるいは同期動作モードのデータを書き込む。※MC8043P/MC8043Pe専用 (データ書き込み命令を実行する)</p> <p>VC void Nmc_WriteData2(int No, int IcNo, int axis, int cmd, long WR6_data, long WR7_data); VB.NET Sub Nmc_WriteData2(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal cmd As Integer, ByVal WR6_data As Integer, ByVal WR7_data As Integer) C# void MC8000P.Nmc_WriteData2(int No, int IcNo, AXIS axis, CMD cmd, int WR6_data, int WR7_data);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照 axis データを書き込む軸。複数軸指定可能。詳細は「4.1.3 補足説明」(2)参照。 cmd データ書き込み命令コード。拡張モードは(60)H, 同期動作モードは(64)Hを指定する。 WR6_data 拡張モードはEM6に, 同期動作モードはSM6に書き込むデータ WR7_data 拡張モードはEM7に, 同期動作モードはSM7に書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 X軸の拡張モードに、EM6データ(5F00)H, EM7データ(45)Hを書き込む。 [VC] Nmc_WriteData2(No, IcNo, AXIS_X, 0x60, 0x5F00, 0x0045); [VB.NET] Call Nmc_WriteData2(No, IcNo, AXIS_X, &H60, &H5F00, &H45) [C#] MC8000P.Nmc_WriteData2(No, IcNo, AXIS.X, 0x60, 0x5F00, 0x0045);</p>
Nmc_ReadData	<p>データ読み出し命令を実行し、データを読み出す。</p> <p>VC long Nmc_ReadData(int No, int IcNo, int axis, int cmd); VB.NET Function Nmc_ReadData(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal cmd As Integer) As Integer C# int MC8000P.Nmc_ReadData(int No, int IcNo, AXIS axis, CMD cmd);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「4.1.3 補足説明」(2)参照。 cmd データ読み出し命令コード((10)H~(14)H)。例: 論理位置カウンタ読み出しは(10)H。 ※MCX304は(14)Hを除く。</p> <p>戻り値 読み出したデータ</p> <p>使用例 [VC] Data = Nmc_ReadData(No, IcNo, AXIS_X, 0x10); // X軸の論理位置カウンタを読み出す。 [VB.NET] Data = Nmc_ReadData(No, IcNo, AXIS_X, &H10) [C#] Data = MC8000P.Nmc_ReadData(No, IcNo, AXIS.X, 0x10);</p>

関数名	機能 及び 内容
Nmc_2BPExec	<p>指定した補間データで2軸ビットパターン補間を実行する。 ※MC8043P/MC8043Pe専用 この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。</p> <p>VC DWORD Nmc_2BPExec(int No, int IcNo, DATA_2BP* pData2Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE);</p> <p>VB.NET Function Nmc_2BPExec(ByVal No As Integer, ByVal IcNo As Integer, ByRef pData2Bp As DATA_2BP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal ContinueFlg As Integer) As Integer</p> <p>C# Nmc_Status MC8000P.Nmc_2BPExec(int No, int IcNo, DATA_2BP[] pData2Bp, int DataCnt, int IpAxis, bool ContinueFlg);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照</p> <p>pData2Bp 2軸BP補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2BPのアドレス)。 DATA_2BPに実行する補間データをセットし、アドレスを指定する。 DATA_2BPについては「4.1.3 補足説明」(3)参照。</p> <p>DataCnt 2軸BP補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。</p> <p>IpAxis 補間を実行する軸。WR5のD0~D5(軸指定)の設定値と同じ値を指定する。 「4.1.3 補足説明」(4)参照。</p> <p>ContinueFlg BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった)場合に 続けるかどうかを設定する。 [VC] TRUE : 続ける、FALSE : 続けない。省略可能。省略時FALSE。 [VB.NET] True : 続ける、False : 続けない [C#] true : 続ける、false : 続けない</p> <p>戻り値</p> <p>補間処理が正常終了した場合は BP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は「4.1.3 補足説明」(1)参照。</p> <p>■正常終了</p> <p>BP_END BP補間処理正常終了</p> <p>■エラーコード</p> <p>BP_CNT_ERR 指定したデータ数が範囲外です</p> <p>BP_ALREADY_EXEC 既にBP補間、あるいは連続補間が実行中です</p> <p>BP_PARAM_ERR 引数の値が正しくない</p> <p>BP_NOT_OPEN_ERR 指定したボードがオープンされていない</p> <p>BP_OTHER_ERR その他のエラー</p> <p>BP_STOP BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった)</p> <p>BP_USER_STOP BP補間実行中にユーザーが中断した</p> <p>BP_DRIVE_ERR BP補間実行中にボードでエラーが発生した(RR0にエラー情報がセットされた)</p> <p>使用例</p> <p>[VC]</p> <pre>// 2軸BP補間データ BP1P, BP1M, BP2P, BP2M DATA_2BP Data2Bp[2] = {{0x0000, 0x2BFF, 0xFFD4, 0x0000}, {0xF6FE, 0x0000, 0x000F, 0x3FC0}};</pre> <p>Nmc_WriteReg5(No, IcNo, 0x04); // 補間軸設定。主軸 : X、第2軸 : Y</p> <p>Ret = Nmc_2BPExec(No, IcNo, Data2Bp, 2, 0x04); // 2軸BP補間実行。データ数2, X, Y軸 if(Ret == BP_END) AfxMessageBox("正常終了"); // 戻り値正常</p> <p>[VB.NET]</p> <pre>Dim Data2Bp(1) As DATA_2BP ' 2軸BP補間データ</pre> <pre>' 2軸BP補間データ設定 Data2Bp(0).Bp1p = &H0: Data2Bp(0).Bp1m = &H2BFF Data2Bp(0).Bp2p = &HFFD4: Data2Bp(0).Bp2m = &H0 Data2Bp(1).Bp1p = &HF6FE: Data2Bp(1).Bp1m = &H0 Data2Bp(1).Bp2p = &HF: Data2Bp(1).Bp2m = &H3FC0</pre>

```

Call Nmc_WriteReg5(No, IcNo, &H4) ' 補間軸設定。主軸：X、第2軸：Y

Ret = Nmc_2BPExec(No, IcNo, Data2Bp(0), 2, &H4, False) ' 2軸BP補間実行。データ数2, X, Y軸

If Ret = BP_END Then ' 戻り値正常
    Call MsgBox("正常終了")
End If

[C#]
DATA_2BP [] Data2Bp = new DATA_2BP[1]; // 2軸BP補間データ
// 補間データ設定
Data2Bp[0].Bp1p = 0xFF30; // 1111 1111 0011 0000 BP1+方向 10パルス
Data2Bp[0].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data2Bp[0].Bp2p = 0; // 0000 0000 0000 0000 BP2+方向 0パルス
Data2Bp[0].Bp2m = 0x84FF; // 1000 0100 1111 1111 BP2-方向 10パルス

IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2; // 補間軸

MC8000P.Nmc_WriteReg5(No, IcNo, IpAxis); // 補間軸設定
MC8000P.Nmc_StartSpd(No, IcNo, AXIS.ALL, 1); // 初速度設定
MC8000P.Nmc_Speed(gBoardNo, IcNo, AXIS.ALL, 1); // ドライブ速度設定

// 2軸BP補間実行
Ret = MC8000P.Nmc_2BPExec(No, IcNo, Data2Bp, DataCnt, Axis, ContinueFlg);

if(Ret == Nmc_Status.BP_END) // 戻り値正常

```

関数名	機能 及び 内容
Nmc_3BPExec	<p>指定した補間データで3軸ビットパターン補間を実行する。 ※MC8043P/MC8043Pe専用 この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。</p> <p>VC DWORD Nmc_3BPExec(int No, int IcNo, DATA_3BP* pData3Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE);</p> <p>VB.NET Function Nmc_3BPExec(ByVal No As Integer, ByVal IcNo As Integer, ByRef pData3Bp As DATA_3BP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal ContinueFlg As Integer) As Integer</p> <p>C# Nmc_Status MC8000P.Nmc_3BPExec(int No, int IcNo, DATA_3BP[] pData3Bp, int DataCnt, int IpAxis, bool ContinueFlg);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照</p> <p>pData3Bp 3軸BP補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_3BPのアドレス)。 DATA_3BPに実行する補間データをセットし、アドレスを指定する。 DATA_3BPについては「4.1.3 補足説明」(3)参照。</p> <p>DataCnt 3軸BP補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。</p> <p>IpAxis 補間を実行する軸。WR5のD0~D5(軸指定)の設定値と同じ値を指定する。 「4.1.3 補足説明」(4)参照。</p> <p>ContinueFlg BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった)場合に 続けるかどうかを設定する。 [VC] TRUE : 続ける、FALSE : 続けない。省略可能。省略時FALSE。 [VB.NET] True : 続ける、False : 続けない [C#] true : 続ける、false : 続けない</p> <p>戻り値</p> <p>補間処理が正常終了した場合は BP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は「4.1.3 補足説明」(1)参照。</p> <p>■正常終了</p> <p>BP_END BP補間処理正常終了</p> <p>■エラーコード</p> <p>BP_CNT_ERR 指定したデータ数が範囲外です</p> <p>BP_ALREADY_EXEC 既にBP補間、あるいは連続補間が実行中です</p> <p>BP_PARAM_ERR 引数の値が正しくない</p> <p>BP_NOT_OPEN_ERR 指定したボードがオープンされていない</p> <p>BP_OTHER_ERR その他のエラー</p> <p>BP_STOP BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった)</p> <p>BP_USER_STOP BP補間実行中にユーザーが中断した</p> <p>BP_DRIVE_ERR BP補間実行中にボードでエラーが発生した(RR0にエラー情報がセットされた)</p> <p>使用例</p> <p>[VC]</p> <pre>// 3軸BP補間データ BP1P, BP1M, BP2P, BP2M, BP3P, BP3M DATA_3BP Data3Bp[2] = {{0xFF30, 0, 0, 0x84FF, 0, 0xAC35}, {0xAC35, 0, 0xC000, 0x36E7, 0xC000, 0x3F3F}};</pre> <p>Nmc_WriteReg5(No, IcNo, 0x24); // 補間軸設定。主軸 : X, 第2軸 : Y, 第3軸 : Z</p> <p>Ret = Nmc_3BPExec(No, IcNo, Data3Bp, 2, 0x24); // 3軸BP補間実行。データ数2, X, Y, Z軸</p> <p>if(Ret == BP_END) AfxMessageBox("正常終了"); // 戻り値正常</p> <p>[VB.NET]</p> <pre>Dim Data3Bp(1) As DATA_3BP ' 3軸BP補間データ ' 3軸BP補間データ設定 Data3Bp(0).Bp1p = &HFF30: Data3Bp(0).Bp1m = &H0 Data3Bp(0).Bp2p = &H0: Data3Bp(0).Bp2m = &H84FF Data3Bp(0).Bp3p = &H0: Data3Bp(0).Bp3m = &HAC35</pre>

```

Data3Bp(1).Bp1p = &HAC35: Data3Bp(1).Bp1m = &H0
Data3Bp(1).Bp2p = &HC000: Data3Bp(1).Bp2m = &H36E7
Data3Bp(1).Bp3p = &HC000: Data3Bp(1).Bp3m = &H3F3F

Call Nmc_WriteReg5(No, IcNo, &H24) ' 補間軸設定。主軸 : X, 第2軸 : Y, 第3軸 : Z

Ret = Nmc_3BPExec(No, IcNo, Data3Bp(0), 2, &H24, False) ' 3軸BP補間実行。データ数2, X,Y,Z軸

If Ret = BP_END Then ' 戻り値正常
    Call MsgBox("正常終了")
End If

[C#]
DATA_3BP [] Data3Bp = new DATA_3BP[1]; // 3軸BP補間データ
// 補間データ設定
Data3Bp[0].Bp1p = 0xFF30; // 1111 1111 0011 0000 BP1+方向 10パルス
Data3Bp[0].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data3Bp[0].Bp2p = 0; // 0000 0000 0000 0000 BP2+方向 0パルス
Data3Bp[0].Bp2m = 0x84FF; // 1000 0100 1111 1111 BP2-方向 10パルス
Data3Bp[0].Bp3p = 0; // 0000 0000 0000 0000 BP3+方向 0パルス
Data3Bp[0].Bp3m = 0xAC35; // 1010 1100 0011 0101 BP3-方向 8パルス

IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2 | IP_AXIS.IP_Z << 4; // 補間軸

MC8000P.Nmc_WriteReg5(No, IcNo, IpAxis); // 補間軸設定
MC8000P.Nmc_StartSpd(No, IcNo, AXIS.ALL, 1); // 初速度設定
MC8000P.Nmc_Speed(No, IcNo, AXIS.ALL, 1); // ドライブ速度設定

// 3軸BP補間実行
Ret = MC8000P.Nmc_3BPExec(No, IcNo, Data3Bp, DataCnt, IpAxis, ContinueFlg);

if(Ret == Nmc_Status.BP_END) // 戻り値正常

```

関数名	機能 及び 内容
Nmc_2BPExec_BG	<p>指定した補間データで2軸ビットパターン補間をバックグラウンドで実行する。 ※MC8043P/MC8043Pe専用 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_BP_ENDメッセージを送信し、終了ステータスを渡します。</p> <p>VC DWORD Nmc_2BPExec_BG(HWND User_hWnd, int No, int IcNo, DATA_2BP* pData2Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE);</p> <p>VB.NET Function Nmc_2BPExec_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByRef pData2Bp As DATA_2BP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal ContinueFlg As Integer) As Integer</p> <p>C# Nmc_Status MC8000P.Nmc_2BPExec_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_2BP[] pData2Bp, int DataCnt, int IpAxis, bool ContinueFlg);</p>
	<p>入カパラメータ</p> <p>User_hWnd ユーザーアプリケーションのウィンドウハンドル No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照 pData2Bp 2軸BP補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2BPのアドレス)。 DATA_2BPに実行する補間データをセットし、アドレスを指定する。 DATA_2BPについては「4.1.3 補足説明」(3)参照。 DataCnt 2軸BP補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis 補間を実行する軸。WR5のD0~D5(軸指定)の設定値と同じ値を指定する。 「4.1.3 補足説明」(4)参照。 ContinueFlg BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった)場合に 続けるかどうかを設定する。 [VC] TRUE : 続ける、FALSE : 続けない。省略可能。省略時FALSE。 [VB.NET] True : 続ける、False : 続けない [C#] true : 続ける、false : 続けない</p>
	<p>戻り値</p> <p>バックグラウンドで補間処理が正常に開始した場合は BP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は「4.1.3 補足説明」(1)参照。</p> <p>■正常開始 BP_START バックグラウンドでBP補間処理が正常に開始した</p> <p>■エラーコード(補間開始前のエラー)</p> <p>BP_CNT_ERR 指定したデータ数が範囲外です BP_ALREADY_EXEC 既にBP補間、あるいは連続補間が実行中です BP_THREAD_ERR スレッドを起動できませんでした BP_MALLOC_ERR メモリを確保できませんでした BP_PARAM_ERR 引数の値が正しくない BP_NOT_OPEN_ERR 指定したボードがオープンされていない BP_OTHER_ERR その他のエラー</p> <p>バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、WM_BP_ENDメッセージを送信します。WM_BP_ENDのメッセージ受信関数で受け取る第1引数にボード番号を第2引数に終了ステータスを渡します。 その終了ステータスは、補間が正常終了した場合は BP_END です。 補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。</p> <p>■正常終了 BP_END BP補間処理正常終了</p> <p>■エラーコード(補間開始後のエラー)</p> <p>BP_STOP BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった) BP_USER_STOP BP補間実行中にユーザーが中断した BP_DRIVE_ERR BP補間実行中にボードでエラーが発生した(RR0にエラー情報がセットされた)</p>

使用例

[VC]

```

{
    // 2軸BP補間データ BP1P, BP1M, BP2P, BP2M
    DATA_2BP Data2Bp[2] = {{0x0000, 0x2BFF, 0xFFD4, 0x0000},
                            {0xF6FE, 0x0000, 0x000F, 0x3FC0}};

    Nmc_WriteReg5(No, IcNo, 0x04); // 補間軸設定。主軸：X、第2軸：Y

    Ret = Nmc_2BPExec_BG(hWnd, No, IcNo, Data2Bp, 2, 0x04); // 2軸BP補間実行。データ数2, X, Y軸

    if(Ret == BP_START) AfxMessageBox("補間開始"); // 戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog) // WM_BP_ENDメッセージ受信関数設定
    ON_MESSAGE(WM_BP_END, OnMsg_BP)
END_MESSAGE_MAP()

// WM_BP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_BP(WPARAM BoardNo, LPARAM Status)
{
    if(Status == BP_END) AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
    return 0;
}

```

[VB.NET]

```

Dim Data2Bp(1) As DATA_2BP ' 2軸BP補間データ

' 2軸BP補間データ設定
Data2Bp(0).Bp1p = &H0: Data2Bp(0).Bp1m = &H2BFF
Data2Bp(0).Bp2p = &HFFD4: Data2Bp(0).Bp2m = &H0
Data2Bp(1).Bp1p = &HF6FE: Data2Bp(1).Bp1m = &H0
Data2Bp(1).Bp2p = &HF: Data2Bp(1).Bp2m = &H3FC0

Call Nmc_WriteReg5(No, IcNo, &H4) ' 補間軸設定。主軸:X, 第2軸:Y

Ret = Nmc_2BPExec_BG(hWnd, No, IcNo, Data2Bp(0), 2, &H4, False) ' 2軸BP補間実行。データ数2, X, Y軸

If Ret = BP_START Then ' 戻り値正常(補間開始)
    Call MsgBox("補間開始")
End If
End Sub

' WM_BP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)

    If m.Msg = WM_BP_END Then ' BP補間終了メッセージ
        If lParam = BP_END Then ' 戻り値正常(補間終了)
            Call MsgBox("補間正常終了")
        End If
    End If

    MyBase.WndProc(m)
End Sub

```

[C#]

```

DATA_2BP [] Data2Bp = new DATA_2BP[1]; // 2軸BP補間データ
// 補間データ設定
Data2Bp[0].Bp1p = 0xFF30; // 1111 1111 0011 0000 BP1+方向 10パルス
Data2Bp[0].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data2Bp[0].Bp2p = 0; // 0000 0000 0000 0000 BP2+方向 0パルス
Data2Bp[0].Bp2m = 0x84FF; // 1000 0100 1111 1111 BP2-方向 10パルス

IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2; // 補間軸

```

```

MC8000P.Nmc_WriteReg5(No, IcNo, IpAxis); // 補間軸設定
MC8000P.Nmc_StartSpd(No, IcNo, AXIS.ALL, 1); // 初速度設定
MC8000P.Nmc_Speed(gBoardNo, IcNo, AXIS.ALL, 1); // ドライブ速度設定

// 2軸BP補間バックグラウンド実行
Ret = MC8000P.Nmc_2BPExec_BG((System.IntPtr)parent.Handle, No, IcNo, Data2Bp,
                             DataCnt, IpAxis, ContinueFlg);
if(Ret == Nmc_Status.BP_START) // 補間開始正常の場合

//WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    // 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
    base.WndProc ( ref m );
    if ( m.Msg == (int)MSG_ID.WM_BP_END )
    {
        if((uint)m.LParam == Nmc_Status.BP_END) // BP補間処理正常終了
    }
}

```

関数名	機能 及び 内容
Nmc_3BPExec_BG	<p>指定した補間データで3軸ビットパターン補間をバックグラウンドで実行する。 ※MC8043P/MC8043Pe専用 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_BP_ENDメッセージを送信し、終了ステータスを渡します。</p> <p>VC DWORD Nmc_3BPExec_BG (HWND User_hWnd, int No, int IcNo, DATA_3BP* pData3Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE);</p> <p>VB.NET Function Nmc_3BPExec_BG (ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByRef pData3Bp As DATA_3BP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal ContinueFlg As Integer) As Integer</p> <p>C# Nmc_Status MC8000P.Nmc_3BPExec_BG (System.IntPtr User_hWnd, int No, int IcNo, DATA_3BP[] pData3Bp, int DataCnt, int IpAxis, bool ContinueFlg);</p> <p>入力パラメータ</p> <p>User_hWnd ユーザーアプリケーションのウィンドウハンドル No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1 詳細は「4.1.3 補足説明」(7)参照 pData3Bp 3軸BP補間データの構造体 (ユーザー定義型) 配列の先頭アドレス (DATA_3BPのアドレス)。 DATA_3BPに実行する補間データをセットし、アドレスを指定する。 DATA_3BPについては「4.1.3 補足説明」(3)参照。 DataCnt 3軸BP補間データの数。構造体 (ユーザー定義型) 配列の配列数を指定する。 IpAxis 補間を実行する軸。WR5のD0~D5 (軸指定) の設定値と同じ値を指定する。 「4.1.3 補足説明」(4)参照。 ContinueFlg BP補間が途中で停止した (速度が速く次データのスタックが間に合わなかった) 場合に 続けるかどうかを設定する。 [VC] TRUE : 続ける、FALSE : 続けない。省略可能。省略時FALSE。 [VB.NET] True : 続ける、False : 続けない [C#] true : 続ける、false : 続けない</p> <p>戻り値</p> <p>バックグラウンドで補間処理が正常に開始した場合は BP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は「4.1.3 補足説明」(1)参照。</p> <p>■ 正常開始 BP_START バックグラウンドでBP補間処理が正常に開始した</p> <p>■ エラーコード (補間開始前のエラー)</p> <p>BP_CNT_ERR 指定したデータ数が範囲外です BP_ALREADY_EXEC 既にBP補間、あるいは連続補間が実行中です BP_THREAD_ERR スレッドを起動できませんでした BP_MALLOC_ERR メモリを確保できませんでした BP_PARAM_ERR 引数の値が正しくない BP_NOT_OPEN_ERR 指定したボードがオープンされていない BP_OTHER_ERR その他のエラー</p> <p>バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、WM_BP_ENDメッセージを送信します。WM_BP_ENDのメッセージ受信関数で受け取る第1引数にボード番号を第2引数に終了ステータスを渡します。 その終了ステータスは、補間が正常終了した場合は BP_END です。 補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。</p> <p>■ 正常終了 BP_END BP補間処理正常終了</p> <p>■ エラーコード (補間開始後のエラー)</p> <p>BP_STOP BP補間が途中で停止した (速度が速く次データのスタックが間に合わなかった) BP_USER_STOP BP補間実行中にユーザーが中断した BP_DRIVE_ERR BP補間実行中にボードでエラーが発生した (RR0にエラー情報がセットされた)</p>

使用例

[VC]

```

{
    // 3軸BP補間データ BP1P, BP1M, BP2P, BP2M, BP3P, BP3M
    DATA_3BP Data3Bp[2] = {{0xFF30, 0, 0, 0x84FF, 0, 0xAC35},
                            {0xAC35, 0, 0xC000, 0x36E7, 0xC000, 0x3F3F}};

    Nmc_WriteReg5(No, IcNo, 0x24); // 補間軸設定。主軸:X, 第2軸:Y, 第3軸:Z

    Ret = Nmc_3BPExec_BG(hWnd, No, IcNo, Data3Bp, 2, 0x24); // 3軸BP補間実行。データ数2, X, Y, Z軸

    if(Ret == BP_START) AfxMessageBox("補間開始"); // 戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog) // WM_BP_ENDメッセージ受信関数設定
    ON_MESSAGE(WM_BP_END, OnMsg_BP)
END_MESSAGE_MAP()

// WM_BP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_BP(WPARAM BoardNo, LPARAM Status)
{
    if(Status == BP_END) AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
    return 0;
}

```

[VB.NET]

```

Dim Data3Bp(1) As DATA_3BP ' 3軸BP補間データ

' 3軸BP補間データ設定
Data3Bp(0).Bp1p = &HFF30: Data3Bp(0).Bp1m = &H0
Data3Bp(0).Bp2p = &H0: Data3Bp(0).Bp2m = &H84FF
Data3Bp(0).Bp3p = &H0: Data3Bp(0).Bp3m = &HAC35
Data3Bp(1).Bp1p = &HAC35: Data3Bp(1).Bp1m = &H0
Data3Bp(1).Bp2p = &HC000: Data3Bp(1).Bp2m = &H36E7
Data3Bp(1).Bp3p = &HC000: Data3Bp(1).Bp3m = &H3F3F

Call Nmc_WriteReg5(No, IcNo, &H24) ' 補間軸設定。主軸:X, 第2軸:Y, 第3軸:Z

Ret = Nmc_3BPExec_BG(hWnd, No, IcNo, Data3Bp(0), 2, &H24, False) ' 3軸BP補間実行。データ数2, X, Y, Z軸

If Ret = BP_START Then ' 戻り値正常(補間開始)
    Call MsgBox("補間開始")
End If
End Sub

' WM_BP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)

    If m.Msg = WM_BP_END Then ' BP補間終了メッセージ
        If lParam = BP_END Then ' 戻り値正常(補間終了)
            Call MsgBox("補間正常終了")
        End If
    End If

    MyBase.WndProc(m)
End Sub

```

[C#]

```

DATA_3BP [] Data3Bp = new DATA_3BP[1]; // 3軸BP補間データ
// 補間データ設定
Data3Bp[0].Bp1p = 0xFF30; // 1111 1111 0011 0000 BP1+方向 10パルス
Data3Bp[0].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data3Bp[0].Bp2p = 0; // 0000 0000 0000 0000 BP2+方向 0パルス
Data3Bp[0].Bp2m = 0x84FF; // 1000 0100 1111 1111 BP2-方向 10パルス
Data3Bp[0].Bp3p = 0; // 0000 0000 0000 0000 BP3+方向 0パルス

```

```

Data3Bp[0].Bp3m = 0xAC35;          // 1010 1100 0011 0101      BP3-方向      8パルス

IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2 | IP_AXIS.IP_Z << 4;      // 補間軸

MC8000P.Nmc_WriteReg5(No, IcNo, IpAxis);          // 補間軸設定
MC8000P.Nmc_StartSpd(No, IcNo, AXIS.ALL, 1);      // 初速度設定
MC8000P.Nmc_Speed(gBoardNo, IcNo, AXIS.ALL, 1);   // ドライブ速度設定

// 3軸BP補間バックグラウンド実行
Ret = MC8000P.Nmc_3BPExec_BG((System.IntPtr)parent.Handle, No, IcNo, Data3Bp,
                             DataCnt, IpAxis, ContinueFlg);

if(Ret == Nmc_Status.BP_START)                    // 補間開始正常の場合

//WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    // 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
    base.WndProc ( ref m );
    if ( m.Msg == (int)MSG_ID.WM_BP_END )
    {
        if((uint)m.LParam == Nmc_Status.BP_END)      // BP補間処理正常終了
        }
    }
}

```

関数名	機能 及び 内容
Nmc_2CIPExec	<p>指定した補間データで2軸連続補間を実行する。 ※MC8043P/MC8043Pe専用</p> <p>この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。</p>
VC	<pre>DWORD Nmc_2CIPExec(int No, int IcNo, DATA_2CIP* pData2Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE);</pre>
VB.NET	<pre>Function Nmc_2CIPExec(ByVal No As Integer, ByVal IcNo As Integer, ByRef pData2Cip As DATA_2CIP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer, ByVal ContinueFlg As Integer) As Integer</pre>
C#	<pre>Nmc_Status MC8000P. Nmc_2CIPExec(int No, int IcNo, DATA_2CIP[] pData2Cip, int DataCnt, int IpAxis, bool SpdChgFlg, bool ContinueFlg);</pre>
入カパラメータ	
No	ボード番号 (ボード上のロータリースイッチの値(0~15))
IcNo	IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照
pData2Cip	2軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2CIPのアドレス)。 DATA_2CIPに実行する補間データをセットし、アドレスを指定する。 DATA_2CIPについては「4.1.3 補足説明」(3)参照。
DataCnt	2軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。
IpAxis	補間を実行する軸。WR5のD0~D5(軸指定)の設定値と同じ値を指定する。 「4.1.3 補足説明」(4)参照。
SpdChgFlg	補間実行中に速度を変更するかどうかを設定する。 変更する場合は「4.1.3 補足説明」(5)参照。 [VC] TRUE : 変更する、FALSE : 変更しない。省略可能。省略時FALSE。 [VB.NET] True : 変更する、False : 変更しない [C#] true : 変更する、false : 変更しない
変更する選択時 : DATA_2CIPのSpeedに設定した値を参照します。 Speedに1~8000の値設定・・・設定した速度に変更します。 Speedに0 設定・・・速度は変更しません。	
変更しない選択時 : DATA_2CIPのSpeedに設定した値を参照しません。	
ContinueFlg	連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった)場合に 続けるかどうかを設定する。 [VC] TRUE : 続ける、FALSE : 続けない。省略可能。省略時FALSE。 [VB.NET] True : 続ける、False : 続けない [C#] true : 続ける、false : 続けない
戻り値	
補間処理が正常終了した場合は CIP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は「4.1.3 補足説明」(1)参照。	
■正常終了	
CIP_END	連続補間処理正常終了
■エラーコード	
CIP_CNT_ERR	指定したデータ数が範囲外です
CIP_ALREADY_EXEC	既にBP補間、あるいは連続補間が実行中です
CIP_CMD_ERR	指定したコマンドが間違っています
CIP_PARAM_ERR	引数の値が正しくない
CIP_NOT_OPEN_ERR	指定したボードがオープンされていない
CIP_OTHER_ERR	その他のエラー
CIP_STOP	連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった)
CIP_USER_STOP	連続補間実行中にユーザーが中断した
CIP_DRIVE_ERR	連続補間実行中にボードでエラーが発生した(RR0にエラー情報がセットされた)
注意	
この関数を実行する前に、初速度を8000に設定して下さい。(本関数実行中に初速度を変更しないで下さい。) 詳細は「4.1.4 使用方法」参照。	

使用例

```
[VC] // 2軸連続補間データ コマンド 速度 終点1 終点2 中心点1 中心点2
DATA_2CIP Data2Cip[2]={ {CMD_IP_2ST, 0, 4500, 0, 0, 0}, // 2軸直線補間
                        {CMD_IP_CCW, 0, 1500, 1500, 0, 1500}}; // CCW円弧補間

Nmc_WriteReg5(No, IcNo, 0x04); // 補間軸設定。主軸：X、第2軸：Y

Ret = Nmc_2CIPExec(No, IcNo, Data2Cip, 2, 0x04); // 2軸連続補間実行。データ数2, X, Y軸

if(Ret == CIP_END) AfxMessageBox("正常終了"); // 戻り値正常

[VB.NET] Dim Data2Cip(1) As DATA_2CIP ' 2軸連続補間データ

' 2軸連続補間データ設定
Data2Cip(0).Command = CMD_IP_2ST ' 2軸直線補間
Data2Cip(0).EndP1 = 4500
Data2Cip(0).EndP2 = 0

Data2Cip(1).Command = CMD_IP_CCW ' CCW円弧補間
Data2Cip(1).EndP1 = 1500
Data2Cip(1).EndP2 = 1500
Data2Cip(1).Center1 = 0
Data2Cip(1).Center2 = 1500

Call Nmc_WriteReg5(No, IcNo, &H4) ' 補間軸設定。主軸：X、第2軸：Y

Ret = Nmc_2CIPExec(No, IcNo, Data2Cip(0), 2, &H4, False, False) ' 2軸連続補間。データ数2, X, Y軸

If Ret = CIP_END Then ' 戻り値正常
    Call MsgBox("正常終了")
End If

[C#]
DATA_2CIP [] Data2Cip = new DATA_2CIP[1]; // 2軸連続補間データ

// 補間データ設定
Data2Cip[0].Command = (ushort)CMD.CMD_IP_CW; // CW円弧補間
Data2Cip[0].EndP1 = 2000;
Data2Cip[0].EndP2 = 2000;
Data2Cip[0].Center1 = 2000;
Data2Cip[0].Center2 = 0;
Data2Cip[0].Speed = 200; // 速度変更する(200)

IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2; // 補間軸

MC8000P.Nmc_WriteReg5(No, IpAxis); // 補間軸設定

// 2軸連続補間実行
// この関数は補間が終了するまで制御が戻りません
Ret = MC8000P.Nmc_2CIPExec(No, int IcNo, Data2Cip, DataCnt, IpAxis, SpdChgFlg, ContinueFlg);

if(Ret == Nmc_Status.CIP_END) // 連続補間処理正常終了
```

関数名	機能 及び 内容
Nmc_3CIPExec	<p>指定した補間データで3軸連続補間を実行する。 ※MC8043P/MC8043Pe専用 この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。</p> <p>VC DWORD Nmc_3CIPExec(int No, int IcNo, DATA_3CIP* pData3Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE);</p> <p>VB.NET Function Nmc_3CIPExec(ByVal No As Integer, ByVal IcNo As Integer, ByRef pData3Cip As DATA_3CIP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer, ByVal ContinueFlg As Integer) As Integer</p> <p>C# Nmc_Status MC8000P.Nmc_3CIPExec(int No, IcNo, DATA_3CIP[] pData3Cip, int DataCnt, int IpAxis, bool SpdChgFlg, bool ContinueFlg);</p> <p>入カパラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照</p> <p>pData3Cip 3軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_3CIPのアドレス)。 DATA_3CIPに実行する補間データをセットし、アドレスを指定する。 DATA_3CIPについては「4.1.3 補足説明」(3)参照。</p> <p>DataCnt 3軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。</p> <p>IpAxis 補間を実行する軸。WR5のD0~D5(軸指定)の設定値と同じ値を指定する。 「4.1.3 補足説明」(4)参照。</p> <p>SpdChgFlg 補間実行中に速度を変更するかどうかを設定する。 変更する場合は「4.1.3 補足説明」(5)参照。 [VC] TRUE : 変更する、FALSE : 変更しない。省略可能。省略時FALSE。 [VB.NET] True : 変更する、False : 変更しない [C#] true : 変更する、false : 変更しない 変更する選択時 : DATA_3CIPのSpeedに設定した値を参照します。 Speedに1~8000の値設定・・・設定した速度に変更します。 Speedに0 設定・・・速度は変更しません。 変更しない選択時 : DATA_3CIPのSpeedに設定した値を参照しません。</p> <p>ContinueFlg 連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった)場合に 続けるかどうかを設定する。 [VC] TRUE : 続ける、FALSE : 続けない。省略可能。省略時FALSE。 [VB.NET] True : 続ける、False : 続けない [C#] true : 続ける、false : 続けない</p> <p>戻り値</p> <p>補間処理が正常終了した場合は CIP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は「4.1.3 補足説明」(1)参照。</p> <p>■ 正常終了 CIP_END 連続補間処理正常終了</p> <p>■ エラーコード</p> <p>CIP_CNT_ERR 指定したデータ数が範囲外です</p> <p>CIP_ALREADY_EXEC 既にBP補間、あるいは連続補間が実行中です</p> <p>CIP_CMD_ERR 指定したコマンドが間違っています</p> <p>CIP_PARAM_ERR 引数の値が正しくない</p> <p>CIP_NOT_OPEN_ERR 指定したボードがオープンされていない</p> <p>CIP_OTHER_ERR その他のエラー</p> <p>CIP_STOP 連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった)</p> <p>CIP_USER_STOP 連続補間実行中にユーザーが中断した</p> <p>CIP_DRIVE_ERR 連続補間実行中にボードでエラーが発生した(RR0にエラー情報がセットされた)</p> <p>注意</p> <p>この関数を実行する前に、初速度を 8000 に設定して下さい。(本関数実行中に初速度を変更しないで下さい。) 詳細は「4.1.4 使用方法」参照。</p>

使用例

```
[VC] DATA_3CIP Data3Cip[2];           // 3軸連続補間データ用

// 3軸連続補間データ設定
Data3Cip[0].EndP1 = 1000;
Data3Cip[0].EndP2 = 2000;
Data3Cip[0].EndP3 = 3000;
Data3Cip[0].Speed = 0;

Data3Cip[1].EndP1 = 2000;
Data3Cip[1].EndP2 = -1000;
Data3Cip[1].EndP3 = 3000;
Data3Cip[1].Speed = 0;

Nmc_WriteReg5(No, IcNo, 0x24);           // 補間軸設定。主軸：X, 第2軸：Y, 第3軸：Z

Ret = Nmc_3CIPExec(No, IcNo, Data3Cip, 2, 0x24); // 3軸連続補間実行。データ数2, X, Y, Z軸

if(Ret == CIP_END)  AfxMessageBox("正常終了"); // 戻り値正常

[VB.NET] Dim Data3Cip(1) As DATA_3CIP    ' 3軸連続補間データ

' 3軸連続補間データ設定
Data3Cip(0).EndP1 = 1000
Data3Cip(0).EndP2 = 2000
Data3Cip(0).EndP3 = 3000
Data3Cip(0).Speed = 0

Data3Cip(1).EndP1 = 2000
Data3Cip(1).EndP2 = -1000
Data3Cip(1).EndP3 = 3000
Data3Cip(1).Speed = 0

Call Nmc_WriteReg5(No, IcNo, &H24)           ' 補間軸設定。主軸：X, 第2軸：Y, 第3軸：Z

Ret = Nmc_3CIPExec(No, IcNo, Data3Cip(0), 2, &H24, False, False) ' 3軸連続補間。データ数2, X, Y, Z軸

If Ret = CIP_END Then                        ' 戻り値正常
    Call MsgBox("正常終了")
End If

[C#]
DATA_3CIP [] Data3Cip = new DATA_3CIP[1];           // 3軸連続補間データ用
// 補間データ設定
Data3Cip[0].EndP1 = 1000;
Data3Cip[0].EndP2 = 2000;
Data3Cip[0].EndP3 = 3000;
Data3Cip[0].Speed = 0;

IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2 | IP_AXIS.IP_Z << 4;           // 補間軸

MC8000P.Nmc_WriteReg5(gBoardNo, IcNo, IpAxis);           // 補間軸設定
MC8000P.Nmc_StartSpd(gBoardNo, IcNo, AXIS.ALL, 8000); // 初速度設定
MC8000P.Nmc_Speed(gBoardNo, IcNo, AXIS.ALL, 100);       // ドライブ速度設定

// 3軸連続補間実行
// この関数は補間が終了するまで制御が戻りません
Ret = MC8000P.Nmc_3CIPExec(No, IcNo, Data3Cip, DataCnt, IpAxis, SpdChgFlg, ContinueFlg);

if(Ret == Nmc_Status.CIP_END)                   // 連続補間処理正常終了
```

関数名	機能 及び 内容
Nmc_2CIPExec_BG	<p>指定した補間データで2軸連続補間をバックグラウンドで実行する。 ※MC8043P/MC8043Pe専用 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_CIP_ENDメッセージを送信し、終了ステータスを渡します。</p>
VC	<p>DWORD Nmc_2CIPExec_BG(HWND User_hWnd, int No, int IcNo, DATA_2CIP* pData2Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE);</p>
VB.NET	<p>Function Nmc_2CIPExec_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByRef pData2Cip As DATA_2CIP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer, ByVal ContinueFlg As Integer) As Integer</p>
C#	<p>Nmc_Status MC8000P.Nmc_2CIPExec_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_2CIP[] pData2Cip, int DataCnt, int IpAxis, bool SpdChgFlg, bool ContinueFlg);</p>
入力パラメータ	
User_hWnd	ユーザーアプリケーションのウィンドウハンドル
No	ボード番号 (ボード上のロータリースイッチの値(0~15))
IcNo	IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照
pData2Cip	2軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2CIPのアドレス)。 DATA_2CIPに実行する補間データをセットし、アドレスを指定する。 DATA_2CIPについては「4.1.3 補足説明」(3)参照。
DataCnt	2軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。
IpAxis	補間を実行する軸。WR5のD0~D5(軸指定)の設定値と同じ値を指定する。 「4.1.3 補足説明」(4)参照。
SpdChgFlg	補間実行中に速度を変更するかどうかを設定する。 変更する場合は「4.1.3 補足説明」(5)参照。 [VC] TRUE : 変更する、FALSE : 変更しない。省略可能。省略時FALSE。 [VB.NET] True : 変更する、False : 変更しない [C#] true : 変更する、false : 変更しない 変更する選択時 : DATA_2CIPのSpeedに設定した値を参照します。 Speedに1~8000の値設定・・・設定した速度に変更します。 Speedに0 設定・・・速度は変更しません。 変更しない選択時 : DATA_2CIPのSpeedに設定した値を参照しません。
ContinueFlg	連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった)場合に続けるかどうかを設定する。 [VC] TRUE : 続ける、FALSE : 続けない。省略可能。省略時FALSE。 [VB.NET] True : 続ける、False : 続けない [C#] true : 続ける、false : 続けない
戻り値	
バックグラウンドで補間処理が正常に開始した場合は CIP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は「4.1.3 補足説明」(1)参照。	
■正常開始	
CIP_START	バックグラウンドで連続補間処理が正常に開始した
■エラーコード(補間開始前のエラー)	
CIP_CNT_ERR	指定したデータ数が範囲外です
CIP_ALREADY_EXEC	既にBP補間、あるいは連続補間が実行中です
CIP_THREAD_ERR	スレッドを起動できませんでした
CIP_MALLOC_ERR	メモリを確保できませんでした
CIP_CMD_ERR	指定したコマンドが間違っています
CIP_PARAM_ERR	引数の値が正しくない
CIP_NOT_OPEN_ERR	指定したボードがオープンされていない
CIP_OTHER_ERR	その他のエラー

バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、WM_CIP_ENDメッセージを送信します。WM_CIP_ENDのメッセージ受信関数で受け取る第1引数にボード番号を、第2引数に終了ステータスを渡します。その終了ステータスは、補間が正常終了した場合は CIP_ENDです。補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。

■正常終了

CIP_END 連続補間処理正常終了

■エラーコード(補間開始後のエラー)

CIP_STOP 連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった)
 CIP_USER_STOP 連続補間実行中にユーザーが中断した
 CIP_DRIVE_ERR 連続補間実行中にボードでエラーが発生した(RR0にエラー情報がセットされた)

注意

この関数を実行する前に、初速度を 8000 に設定して下さい。(本関数実行中に初速度を変更しないで下さい。) 詳細は「4.1.4 使用方法」参照。

使用例

[VC]

```
{
// 2軸連続補間データ コマンド 速度 終点1 終点2 中心点1 中心点2
DATA_2CIP Data2Cip[2]={{CMD_IP_2ST, 0, 4500, 0, 0, 0}, // 2軸直線補間
{CMD_IP_CCW, 0, 1500, 1500, 0, 1500}}; // CCW円弧補間

Nmc_WriteReg5(No, IcNo, 0x04); // 補間軸設定。主軸：X、第2軸：Y

Ret = Nmc_2CIPExec_BG(hWnd, No, IcNo, Data2Cip, 2, 0x04); // 2軸連続補間実行。データ数2, X, Y軸

if(Ret == CIP_START) AfxMessageBox("補間開始"); // 戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog) // WM_CIP_ENDメッセージ受信関数設定
ON_MESSAGE(WM_CIP_END, OnMsg_CIP)
END_MESSAGE_MAP()

// WM_CIP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_CIP(WPARAM BoardNo, LPARAM Status)
{
if(Status == CIP_END) AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
return 0;
}
```

[VB.NET]

```
Dim Data2Cip(1) As DATA_2CIP ' 2軸連続補間データ

' 2軸連続補間データ設定
Data2Cip(0).Command = CMD_IP_2ST ' 2軸直線補間
Data2Cip(0).EndP1 = 4500
Data2Cip(0).EndP2 = 0

Data2Cip(1).Command = CMD_IP_CCW ' CCW円弧補間
Data2Cip(1).EndP1 = 1500
Data2Cip(1).EndP2 = 1500
Data2Cip(1).Center1 = 0
Data2Cip(1).Center2 = 1500

Call Nmc_WriteReg5(No, IcNo, &H4) ' 補間軸設定。主軸：X, 第2軸：Y
' 2軸連続補間。データ数2, X, Y軸

Ret = Nmc_2CIPExec_BG(hWnd, No, IcNo, Data2Cip(0), 2, &H4, False, False)

If Ret = CIP_START Then ' 戻り値正常(補間開始)
Call MsgBox("補間開始")
End If
End Sub
```

```

'WM_CIP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)
    If m.Msg = WM_CIP_END Then          '連続補間終了メッセージ
        If lParam = CIP_END Then        '戻り値正常(補間終了)
            Call MsgBox("補間正常終了")
        End If
    End If
    MyBase.WndProc(m)
End Sub

[C#]
DATA_2CIP [] Data2Cip = new DATA_2CIP[1];    // 2軸連続補間データ

// 補間データ設定
Data2Cip[0].Command = (ushort)CMD.CMD_IP_CW; // CW円弧補間
Data2Cip[0].EndP1 = 2000;
Data2Cip[0].EndP2 = 2000;
Data2Cip[0].Center1 = 2000;
Data2Cip[0].Center2 = 0;
Data2Cip[0].Speed = 200;                    // 速度変更する(200)

IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2; // 補間軸

MC8000P.Nmc_WriteReg5(No, int IcNo, IpAxis); // 補間軸設定

// 2軸連続補間実行
// バックグラウンドで実行する
Ret = MC8000P.Nmc_2CIPExec_BG((System.IntPtr)parent.Handle, dNo, int IcNo, Data2Cip,
                                DataCnt, IpAxis, SpdChgFlg, ContinueFlg);

if(Ret == Nmc_Status.CIP_START)             // 連続補間処理正常開始

//WM_CIP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    // 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
    base.WndProc ( ref m );
    if ( m.Msg == (int)MSG_ID.WM_CIP_END )
    {
        if((uint)m.LParam == Nmc_Status.CIP_END) // 連続補間処理正常終了
    }
}

```


■正常終了
CIP_END 連続補間処理正常終了

■エラーコード(補間開始後のエラー)

CIP_STOP 連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった)
CIP_USER_STOP 連続補間実行中にユーザーが中断した
CIP_DRIVE_ERR 連続補間実行中にボードでエラーが発生した(RR0にエラー情報がセットされた)

注意

この関数を実行する前に、初速度を 8000 に設定して下さい。(本関数実行中に初速度を変更しないで下さい。)
詳細は「4.1.4 使用方法」参照。

使用例

[VC]

```
{
    DATA_3CIP Data3Cip[2];          // 3軸連続補間データ用

    // 3軸連続補間データ設定
    Data3Cip[0].EndP1 = 1000;
    Data3Cip[0].EndP2 = 2000;
    Data3Cip[0].EndP3 = 3000;

    Data3Cip[1].EndP1 = 2000;
    Data3Cip[1].EndP2 = -1000;
    Data3Cip[1].EndP3 = 3000;

    Nmc_WriteReg5(No, IcNo, 0x24);          // 補間軸設定。主軸：X, 第2軸：Y, 第3軸：Z

    Ret = Nmc_3CIPExec_BG(hWnd, No, IcNo, Data3Cip, 2, 0x24); // 3軸連続補間実行。データ数2, X, Y, Z軸
    if(Ret == CIP_START)  AfxMessageBox("補間開始"); // 戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog) // WM_CIP_ENDメッセージ受信関数設定
    ON_MESSAGE(WM_CIP_END, OnMsg_CIP)
END_MESSAGE_MAP()

// WM_CIP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_CIP(WPARAM BoardNo, LPARAM Status)
{
    if(Status == CIP_END)  AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
    return 0;
}
```

[VB.NET]

```
Dim Data3Cip(1) As DATA_3CIP ' 3軸連続補間データ

' 3軸連続補間データ設定
Data3Cip(0).EndP1 = 1000
Data3Cip(0).EndP2 = 2000
Data3Cip(0).EndP3 = 3000

Data3Cip(1).EndP1 = 2000
Data3Cip(1).EndP2 = -1000
Data3Cip(1).EndP3 = 3000

Call Nmc_WriteReg5(No, IcNo, &H24) ' 補間軸設定。主軸：X, 第2軸：Y, 第3軸：Z
' 3軸連続補間。データ数2, X, Y, Z軸

Ret = Nmc_3CIPExec_BG(hWnd, No, IcNo, Data3Cip(0), 2, &H24, False, False)

If Ret = CIP_START Then ' 戻り値正常(補間開始)
    Call MsgBox("補間開始")
End If
End Sub
```

```

'WM_CIP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)

    If m.Msg = WM_CIP_END Then          '連続補間終了メッセージ
        If lParam = CIP_END Then      '戻り値正常(補間終了)
            Call MsgBox("補間正常終了")
        End If
    End If

    MyBase.WndProc(m)

End Sub

[C#]
DATA_3CIP [] Data3Cip = new DATA_3CIP[1];          // 3軸連続補間データ用
// 補間データ設定
Data3Cip[0].EndP1 = 1000;
Data3Cip[0].EndP2 = 2000;
Data3Cip[0].EndP3 = 3000;
Data3Cip[0].Speed = 0;

IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y << 2 | IP_AXIS.IP_Z << 4;    // 補間軸

MC8000P.Nmc_WriteReg5(gBoardNo, int IcNo, IpAxis);          // 補間軸設定
MC8000P.Nmc_StartSpd(gBoardNo, int IcNo, AXIS.ALL, 8000);   // 初速度設定
MC8000P.Nmc_Speed(gBoardNo, int IcNo, AXIS.ALL, 100);       // ドライブ速度設定

// 3軸連続補間実行
// バックグラウンドで実行する
Ret = MC8000P.Nmc_3CIPExec_BG((System.IntPtr)parent.Handle, gBoardNo, int IcNo,
                               Data3Cip, DataCnt, IpAxis, SpdChgFlg, ContinueFlg);
if(Ret == Nmc_Status.CIP_START)                             // 連続補間処理正常開始

//WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    // 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
    base.WndProc ( ref m );
    if ( m.Msg == (int)MSG_ID.WM_CIP_END )
    {
        if((uint)m.LParam == Nmc_Status.CIP_END)             // 連続補間処理正常終了
    }
}

```

関数名	機能 及び 内容
Nmc_IPStop	<p>補間処理を中断する。 ※MC8043P/MC8043Pe専用 補間ドライブを即停止し、Nmc_xxxの補間関数で実行中の補間処理を終了します。</p> <p>Nmc_IPStopを使用して補間処理を中断した場合、各補間関数の戻り値は下記のエラーコードです。 ◆BP補間 : BP_USER_STOP ◆連続補間 : CIP_USER_STOP</p> <p>VC BOOL Nmc_IPStop(int No, int IcNo); VB.NET Function Nmc_IPStop(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# bool MC8000P.Nmc_IPStop(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「4.1.3 補足説明」(7)参照。</p> <p>戻り値 [VC] 補間処理の中断に成功するとTRUE、失敗するとFALSE [VB.NET] 補間処理の中断に成功すると0以外、失敗すると0 [C#] 補間処理の中断に成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] Nmc_IPStop(No, IcNo); // 補間処理を中断する [VB.NET] Call Nmc_IPStop(No, IcNo) [C#] MC8000P.Nmc_IPStop(No, IcNo);</p>
Nmc_IPGetMsgNo	<p>補間終了時に受信した下記メッセージの引数wParamからボード番号とIC番号を取得する。 ※MC8043P/MC8043Pe専用</p> <p>◆BP補間 : WM_BP_END ◆連続補間 : WM_CIP_END</p> <p>VC void Nmc_IPGetMsgNo(int wParam, int* No, int* IcNo); VB.NET Sub Nmc_IPGetMsgNo(ByVal wParam As Integer, ByRef No As Integer, ByRef IcNo As Integer) C# bool MC8000P.Nmc_IPGetMsgNo(int wParam, out int No, out int IcNo)</p> <p>入力パラメータ wParam 補間終了時に受信したメッセージの引数wParam No [VC] ボード番号を格納する変数のアドレス。 [VB.NET][C#] ボード番号を格納する変数。 IcNo [VC] IC番号を格納する変数のアドレス。 [VB.NET][C#] IC番号を格納する変数。</p> <p>戻り値 なし</p> <p>使用例 [VC] int BdNo; // ボード番号 int IcNo; // IC番号 Nmc_IPGetMsgNo(wParam, &BdNo, &IcNo);</p> <p>[VB.NET] Dim BdNo As Integer ' ボード番号 Dim IcNo As Integer ' IC番号 Call Nmc_IPGetMsgNo(m.WParam.ToInt32(), BdNo, IcNo)</p> <p>[C#] int BdNo; // ボード番号 int IcNo; // IC番号 (0~1) MC8000P.Nmc_IPGetMsgNo(WParam, out BdNo, out IcNo)</p>

4.1.3 補足説明

(1) 各定義は、下記のファイルで行っています。

```
VC・・・ MC8000P_DLL.H  
VB.NET・・・MC8000P_DLL.vb  
C#・・・入力支援により各定義を参照、引用できます。
```

VC++、VB.NET、C#の定義内容を以下に示します。

①レジスタ番号

[VC]

```
#define MCX_WR0      0x0000 // WR 0  
#define MCX_WR1      0x0001 // WR 1  
#define MCX_WR2      0x0002 // WR 2  
#define MCX_WR3      0x0003 // WR 3  
#define MCX_WR4      0x0004 // WR 4  
#define MCX_WR5      0x0005 // WR 5  
#define MCX_WR6      0x0006 // WR 6  
#define MCX_WR7      0x0007 // WR 7  
  
#define MCX_RR0      0x0000 // RR 0  
#define MCX_RR1      0x0001 // RR 1  
#define MCX_RR2      0x0002 // RR 2  
#define MCX_RR3      0x0003 // RR 3  
#define MCX_RR4      0x0004 // RR 4  
#define MCX_RR5      0x0005 // RR 5  
#define MCX_RR6      0x0006 // RR 6  
#define MCX_RR7      0x0007 // RR 7
```

[C#]

```
// ■Nmc_OutPort/Nmc_InPort用  
// 搭載ICが1つのとき WR0_A ~ WR7_A/RR0_A ~ RR7_Aを使用  
// 搭載ICが2つのとき IC_AにはWR0_A ~ WR7_A/RR0_A ~ RR7_Aを使用  
//                               IC_BにはWR0_B ~ WR7_B/RR0_B ~ RR7_Bを使用  
// ■Nmc_WriteReg/Nmc_ReadReg用  
// WR0 ~ WR7/RR0 ~ RR7を使用  
  
public enum REG_MCX : int  
{  
    //■Nmc_OutPort用  
    // ライトレジスタのアドレス  
    // IC-AのWR0~WR7  
    WR0_A =0x0000,  
    WR1_A =0x0001,  
    WR2_A =0x0002,  
    WR3_A =0x0003,  
    WR4_A =0x0004,  
    WR5_A =0x0005,  
    WR6_A =0x0006,  
    WR7_A =0x0007,  
  
    // IC-BのWR0~WR7  
    WR0_B =0x0008,  
    WR1_B =0x0009,  
    WR2_B =0x000A,  
    WR3_B =0x000B,  
    WR4_B =0x000C,  
    WR5_B =0x000D,  
    WR6_B =0x000E,  
    WR7_B =0x000F,
```

```

// PIX132のアドレス
WR14  =0x0014,
WR15  =0x0015,
WR16  =0x0016,

//■Nmc_InPort用
// リードレジスタのアドレス
// IC-AのRR0~RR7
RR0_A  =0x0000,
RR1_A  =0x0001,
RR2_A  =0x0002,
RR3_A  =0x0003,
RR4_A  =0x0004,
RR5_A  =0x0005,
RR6_A  =0x0006,
RR7_A  =0x0007,

// IC-BのRR0~RR7
RR0_B  =0x0008,
RR1_B  =0x0009,
RR2_B  =0x000A,
RR3_B  =0x000B,
RR4_B  =0x000C,
RR5_B  =0x000D,
RR6_B  =0x000E,
RR7_B  =0x000F,

// PIX132のアドレス
RR14  =0x0014,
RR15  =0x0015,
RR16  =0x0016,

//■Nmc_WriteReg用
// ライトレジスタのアドレス
WR0    =0x0000,
WR1    =0x0001,
WR2    =0x0002,
WR3    =0x0003,
WR4    =0x0004,
WR5    =0x0005,
WR6    =0x0006,
WR7    =0x0007,

//■Nmc_ReadReg用
// リードレジスタのアドレス
RR0    =0x0000,
RR1    =0x0001,
RR2    =0x0002,
RR3    =0x0003,
RR4    =0x0004,
RR5    =0x0005,
RR6    =0x0006,
RR7    =0x0007
}
(使用例) REG_MCXのWR0の場合は      REG_MCX.WR0

```

②軸定義

[VC]

```

#define AXIS_ALL      0xF // 全軸
#define AXIS_X        0x1 // X軸
#define AXIS_Y        0x2 // Y軸
#define AXIS_Z        0x4 // Z軸
#define AXIS_U        0x8 // U軸
#define AXIS_NONE     0   // 軸指定無し

```

```
[C#]
public enum AXIS : int
{
    // 軸定義
    ALL           =0xF, // 全軸
    X             =0x1, // X軸
    Y             =0x2, // Y軸
    Z             =0x4, // Z軸
    U             =0x8, // U軸
    NONE          =0,   // 軸指定無し
}
(使用例) 全軸の場合は      AXIS.ALL
```

③デバイスID

デバイスIDとは、ポートに割り当てられたの固有の番号です。
各ボードの番号は以下の通りです。

ボード	デバイスID
MC8043P / MC8043Pe	0xA0A2
MC8082P / MC8082Pe	0xA0D0

(注) MC8022P、MC8042PはMC8082PのデバイスIDと同じです。

```
[VC]
#define ID_MC8043P      0xA0A2 // MC8043P及びMC8043Pe
#define ID_MC8082P      0xA0D0 // MC8082P、MC8022P、MC8042P及びMC8082Pe
(使用例) MC8082P、42P、22P及びMC8082PeはID_MC8082P、MC8043P及びMC8082PeはID_MC8043P
```

```
[C#]
public enum Dev_ID : ushort
{
    MC8043P      =0xA0A2, // MC8043P及びMC8043Pe
    MC8082P      =0xA0D0 // MC8082P、42P、22P及びMC8082Pe
}
(使用例) MC8082P、42P、22P及びMC8082PeはDev_ID.MC8082P、MC8043P及びMC8043PeはDev_ID.MC8043P
```

④コマンド定義 ※使用できるコマンドは各ICの取扱説明書参照。 (I)と(II)のコマンドはどちらを使用しても構いません。

```
[VC]
// ドライブ命令
//(I)
#define CMD_F_DRV_P      0x20 // +方向定量パルスドライブ
#define CMD_F_DRV_M      0x21 // -方向定量パルスドライブ
#define CMD_C_DRV_P      0x22 // +方向連続パルスドライブ
#define CMD_C_DRV_M      0x23 // -方向連続パルスドライブ
#define CMD_START_HOLD   0x24 // ドライブ開始ホールド
#define CMD_START_FREE   0x25 // ドライブ開始フリー/終了ステータスクリア
#define CMD_STP_STS_CLR  0x25 // ドライブ開始フリー/終了ステータスクリア
#define CMD_STOP_DEC     0x26 // ドライブ減速停止
#define CMD_STOP_SUDDEN  0x27 // ドライブ即停止

//(II)
#define MC8000P_CMD_DRVFP  CMD_F_DRV_P // +方向定量パルスドライブ
#define MC8000P_CMD_DRVFM  CMD_F_DRV_M // -方向定量パルスドライブ
#define MC8000P_CMD_DRVVP  CMD_C_DRV_P // +方向連続パルスドライブ
#define MC8000P_CMD_DRVVM  CMD_C_DRV_M // -方向連続パルスドライブ
#define MC8000P_CMD_DHOLD  CMD_START_HOLD // ドライブ開始ホールド
#define MC8000P_CMD_DFREF  CMD_START_FREE // ドライブ開始フリー
#define MC8000P_CMD_STSCLR  CMD_STP_STS_CLR // 終了ステータスクリア
#define MC8000P_CMD_DRVSBRK  CMD_STOP_DEC // ドライブ減速停止
#define MC8000P_CMD_DRVFBRK  CMD_STOP_SUDDEN // ドライブ即停止

// 補間命令 ※MC8043P/MC8043Pe専用
//(I)
#define CMD_IP_2ST      0x30 // 2軸直線補間ドライブ
#define CMD_IP_3ST      0x31 // 3軸直線補間ドライブ
#define CMD_IP_CW      0x32 // CW円弧補間ドライブ
#define CMD_IP_CCW      0x33 // CCW円弧補間ドライブ
```

```

#define CMD_IP_2BP          0x34          // 2軸ビットパターン補間ドライブ
#define CMD_IP_3BP          0x35          // 3軸ビットパターン補間ドライブ
#define CMD_BP_ENABLED      0x36          // B Pレジスタ書き込み可
#define CMD_BP_DISABLED     0x37          // B Pレジスタ書き込み不可
#define CMD_BP_STACK        0x38          // B Pデータスタック
#define CMD_BP_CLR          0x39          // B Pデータクリア
#define CMD_IP_1STEP        0x3A          // 補間シングルステップ
#define CMD_IP_DEC_VALID    0x3B          // 減速有効
#define CMD_IP_DEC_INVALID  0x3C          // 減速無効
#define CMD_IP_INTRPT_CLR   0x3D          // 補間割り込みクリア
(Ⅱ)
#define MC8000P_CMD_2CIP    CMD_IP_2ST   // 2軸直線補間ドライブ
#define MC8000P_CMD_3CIP    CMD_IP_3ST   // 3軸直線補間ドライブ
#define MC8000P_CMD_CIPCW   CMD_IP_CW    // CW円弧補間ドライブ
#define MC8000P_CMD_CIPCCW  CMD_IP_CCW   // CCW円弧補間ドライブ
#define MC8000P_CMD_BPIP2   CMD_IP_2BP   // 2軸ビットパターン補間ドライブ
#define MC8000P_CMD_BPIP3   CMD_IP_3BP   // 3軸ビットパターン補間ドライブ
#define MC8000P_CMD_BPENABLED  CMD_BP_ENABLED // B Pレジスタ書き込み可
#define MC8000P_CMD_BPDISABLED  CMD_BP_DISABLED // B Pレジスタ書き込み不可
#define MC8000P_CMD_BPSTACK    CMD_BP_STACK // B Pデータスタック
#define MC8000P_CMD_BPCLR      CMD_BP_CLR // B Pデータクリア
#define MC8000P_CMD_IP1STEP    CMD_IP_1STEP // 補間シングルステップ
#define MC8000P_CMD_DECEN      CMD_IP_DEC_VALID // 減速有効
#define MC8000P_CMD_DECDIS     CMD_IP_DEC_INVALID // 減速無効
#define MC8000P_CMD_CLRINTRPT  CMD_IP_INTRPT_CLR // 補間割り込みクリア

// その他の命令
//(Ⅰ)
#define CMD_HOME_EXEC        0x62          // 自動原点出し実行
#define CMD_DEVCTR_CLR       0x63          // 偏差カウンタクリアパルス出力
#define CMD_SYNC_ACTIVE      0x65          // 同期動作起動 ※MCX314As 専用
#define CMD_NOP               0x0F          // NOP (軸切り換え用)
//(Ⅱ)
#define MC8000P_CMD_HMSRC    CMD_HOME_EXEC // 自動原点出し実行
#define MC8000P_CMD_DCC      CMD_DEVCTR_CLR // 偏差カウンタクリアパルス出力
#define MC8000P_CMD_SYNCACT  CMD_SYNC_ACTIVE // 同期動作起動 ※MCX314As 専用
#define MC8000P_CMD_NOP      CMD_NOP       // NOP (軸切り換え用)

```

[C#]

```
public enum CMD : int
```

```
{
```

```

// ドライブ命令
//(Ⅰ)
CMD_F_DRV_P          = 0x20,          // +方向定量パルスドライブ
CMD_F_DRV_M          = 0x21,          // -方向定量パルスドライブ
CMD_C_DRV_P          = 0x22,          // +方向連続パルスドライブ
CMD_C_DRV_M          = 0x23,          // -方向連続パルスドライブ
CMD_START_HOLD       = 0x24,          // ドライブ開始ホールド
CMD_START_FREE       = 0x25,          // ドライブ開始フリー/終了ステータスクリア
CMD_STP_STS_CLR      = 0x25,          // ドライブ開始フリー/終了ステータスクリア
CMD_STOP_DEC         = 0x26,          // ドライブ減速停止
CMD_STOP_SUDDEN      = 0x27,          // ドライブ即停止
//(Ⅱ)
MC8000P_CMD_DRVFP    = CMD_F_DRV_P,   // +方向定量パルスドライブ
MC8000P_CMD_DRVFM    = CMD_F_DRV_M,   // -方向定量パルスドライブ
MC8000P_CMD_DRVVP    = CMD_C_DRV_P,   // +方向連続パルスドライブ
MC8000P_CMD_DRVVM    = CMD_C_DRV_M,   // -方向連続パルスドライブ
MC8000P_CMD_DHOLD    = CMD_START_HOLD, // ドライブ開始ホールド
MC8000P_CMD_DFREET   = CMD_START_FREE, // ドライブ開始フリー
MC8000P_CMD_STSCLR   = CMD_STP_STS_CLR, // 終了ステータスクリア
MC8000P_CMD_DRVSBRK  = CMD_STOP_DEC,   // ドライブ減速停止
MC8000P_CMD_DRVFBRK  = CMD_STOP_SUDDEN, // ドライブ即停止

```

```

// 補間命令      ※MC8043P/MC8043Pe専用
//( I )
CMD_IP_2ST      = 0x30,          // 2軸直線補間ドライブ
CMD_IP_3ST      = 0x31,          // 3軸直線補間ドライブ
CMD_IP_CW       = 0x32,          // CW円弧補間ドライブ
CMD_IP_CCW      = 0x33,          // CCW円弧補間ドライブ
CMD_IP_2BP      = 0x34,          // 2軸ビットパターン補間ドライブ
CMD_IP_3BP      = 0x35,          // 3軸ビットパターン補間ドライブ
CMD_BP_ENABLED  = 0x36,          // BPレジスタ書き込み可
CMD_BP_DISABLED = 0x37,          // BPレジスタ書き込み不可
CMD_BP_STACK    = 0x38,          // BPデータスタック
CMD_BP_CLR      = 0x39,          // BPデータクリア
CMD_IP_1STEP    = 0x3A,          // 補間シングルステップ
CMD_IP_DEC_VALID = 0x3B,          // 減速有効
CMD_IP_DEC_INVALID = 0x3C,       // 減速無効
CMD_IP_INTRPT_CLR = 0x3D,       // 補間割り込みクリア

//( II )
MC8000P_CMD_2CIP = CMD_IP_2ST,   // 2軸直線補間ドライブ
MC8000P_CMD_3CIP = CMD_IP_3ST,   // 3軸直線補間ドライブ
MC8000P_CMD_CIPCW = CMD_IP_CW,    // CW円弧補間ドライブ
MC8000P_CMD_CIPCCW = CMD_IP_CCW,  // CCW円弧補間ドライブ
MC8000P_CMD_BP_IP2 = CMD_IP_2BP,  // 2軸ビットパターン補間ドライブ
MC8000P_CMD_BP_IP3 = CMD_IP_3BP,  // 3軸ビットパターン補間ドライブ
MC8000P_CMD_BP_ENABLED = CMD_BP_ENABLED, // BPレジスタ書き込み可
MC8000P_CMD_BP_DISABLED = CMD_BP_DISABLED, // BPレジスタ書き込み不可
MC8000P_CMD_BP_STACK = CMD_BP_STACK, // BPデータスタック
MC8000P_CMD_BP_CLR = CMD_BP_CLR,  // BPデータクリア
MC8000P_CMD_IP_1STEP = CMD_IP_1STEP, // 補間シングルステップ
MC8000P_CMD_IP_DECEN = CMD_IP_DEC_VALID, // 減速有効
MC8000P_CMD_IP_DECDIS = CMD_IP_DEC_INVALID, // 減速無効
MC8000P_CMD_IP_INTRPT = CMD_IP_INTRPT_CLR, // 補間割り込みクリア

// その他の命令
//( I )
CMD_HOME_EXEC    = 0x62,          // 自動原点出し実行
CMD_DEVCTR_CLR   = 0x63,          // 偏差カウンタクリアパルス出力
CMD_SYNC_ACTIVE  = 0x65,          // 同期動作起動      ※MCX314As 専用

//( II )
MC8000P_CMD_HMSRC = CMD_HOME_EXEC, // 自動原点出し実行
MC8000P_CMD_DCC   = CMD_DEVCTR_CLR, // 偏差カウンタクリアパルス出力
MC8000P_CMD_SYNCACT = CMD_SYNC_ACTIVE, // 同期動作起動      ※MCX314As 専用
MC8000P_CMD_NOP   = CMD_NOP,      // NOP (軸切り換え用)
}

```

(使用例) 2軸直線補間ドライブを指定する場合は CMD. CMD_IP_2ST

⑤補間終了メッセージ、終了ステータス ※MC8043P/MC8043Pe専用

[VC]

```

// 補間終了メッセージ
#define WM_BP_END          (WM_USER + 1)      // BP補間終了メッセージ
#define WM_CIP_END        (WM_USER + 2)      // 連続補間終了メッセージ

//***** BP補間 終了ステータス *****
// ■正常
#define BP_START          0x101              // バックグラウンドでBP補間を開始した
#define BP_END            0x102              // BP補間正常終了

// ■補間開始前のエラー
#define BP_CNT_ERR        0x111              // 指定されたデータ数が範囲外
#define BP_ALREADY_EXEC   0x112              // 既にBP補間、あるいは連続補間が実行中
#define BP_THREAD_ERR     0x113              // スレッドを起動できなかった
#define BP_MALLOC_ERR     0x114              // メモリを確保できなかった
#define BP_PARAM_ERR      0x116              // 引数の値が正しくない
#define BP_NOT_OPEN_ERR   0x117              // 指定したボードがオープンされていない

```

```

#define BP_OTHER_ERR          0x118          // その他のエラー

// ■補間実行中のエラー
#define BP_STOP              0x121          // BP補間が途中で停止した
// (速度が速く次データのスタックが間に合わなかった場合)
#define BP_USER_STOP        0x122          // BP補間実行中にユーザーが中断した
#define BP_DRIVE_ERR        0x123          // BP補間実行中にボードでエラー発生
// (RR0にエラー情報がセットされた)

//***** 連続補間 終了ステータス *****
// ■正常
#define CIP_START            0x201          // バックグラウンドで連続補間を開始した
#define CIP_END              0x202          // 連続補間正常終了

// ■補間開始前のエラー
#define CIP_CNT_ERR          0x211          // 指定されたデータ数が範囲外
#define CIP_ALREADY_EXEC    0x212          // 既にB P補間、あるいは連続補間が実行中
#define CIP_THREAD_ERR       0x213          // スレッドを起動できなかった
#define CIP_MALLOC_ERR       0x214          // メモリを確保できなかった
#define CIP_CMD_ERR          0x215          // コマンドエラー (ユーザーが指定したコマンドが間違っている)
#define CIP_PARAM_ERR        0x216          // 引数の値が正しくない
#define CIP_NOT_OPEN_ERR     0x217          // 指定したボードがオープンされていない
#define CIP_OTHER_ERR        0x218          // その他のエラー

// ■補間実行中のエラー
#define CIP_STOP             0x221          // 連続補間が途中で停止した
// (速度が速く次データのセットが間に合わなかった場合)
#define CIP_USER_STOP        0x222          // 連続補間実行中にユーザーが中断した
#define CIP_DRIVE_ERR        0x223          // 連続補間実行中にボードでエラー発生
// (RR0にエラー情報がセットされた)

```

[C#]

```

public enum MSG_ID : int
{
    // 補間終了メッセージ
    WM_USER          = 0x0400,
    WM_BP_END        = WM_USER + 1, // (WM_USER + 1) B P補間終了メッセージ
    WM_CIP_END        = WM_USER + 2, // (WM_USER + 2) 連続補間終了メッセージ
}
(使用例) MSG_IDのWM_BP_ENDの場合は      MSG_ID.WM_BP_END
public enum Nmc_Status : uint
{
    //***** BP補間 終了ステータス *****
    // ■正常
    BP_START          = 0x101,          // バックグラウンドでBP補間を開始した
    BP_END            = 0x102,          // BP補間正常終了

    // ■補間開始前のエラー
    BP_CNT_ERR        = 0x111,          // 指定されたデータ数が範囲外
    BP_ALREADY_EXEC   = 0x112,          // 既にB P補間、あるいは連続補間が実行中
    BP_THREAD_ERR     = 0x113,          // スレッドを起動できなかった
    BP_MALLOC_ERR     = 0x114,          // メモリを確保できなかった
    // ■補間実行中のエラー
    BP_STOP           = 0x121,          // BP補間が途中で停止した
// (速度が速く次データのスタックが間に合わなかった場合)
    BP_USER_STOP      = 0x122,          // BP補間実行中にユーザーが中断した
    BP_DRIVE_ERR      = 0x123,          // BP補間実行中にボードでエラー発生
// (RR0にエラー情報がセットされた)

    //***** 連続補間 終了ステータス *****
    // ■正常
    CIP_START          = 0x201,          // バックグラウンドで連続補間を開始した
    CIP_END            = 0x202,          // 連続補間正常終了
}

```

```

// ■補間開始前のエラー
CIP_CNT_ERR           = 0x211,      // 指定されたデータ数が範囲外
CIP_ALREADY_EXEC     = 0x212,      // 既にB P補間、あるいは連続補間が実行中
CIP_THREAD_ERR       = 0x213,      // スレッドを起動できなかった
CIP_MALLOC_ERR       = 0x214,      // メモリを確保できなかった
CIP_CMD_ERR          = 0x215,      // コマンドエラー（ユーザーが指定したコマンドが間違っている）

// ■補間実行中のエラー
CIP_STOP              = 0x221,      // 連続補間が途中で停止した
                                   // （速度が速く次データのセットが間に合わなかった場合）
CIP_USER_STOP        = 0x222,      // 連続補間実行中にユーザーが中断した
CIP_DRIVE_ERR        = 0x223,      // 連続補間実行中にボードでエラー発生
                                   // （RR0にエラー情報がセットされた）
}

```

（使用例）バックグラウンドでBP補間を開始した場合は `Nmc_Status.BP_START`

```

public enum IP_AXIS : int
{
    // 補間軸
    IP_X           =0,      // 補間軸 X
    IP_Y           =1,      // 補間軸 Y
    IP_Z           =2,      // 補間軸 Z
    IP_U           =3,      // 補間軸 U
}

```

（使用例）補間軸 Xを指定する場合は `IP_AXIS.IP_X`

⑥定数定義

[C#]

```

public enum CONST : int
{
    MAX_BOARD_MC8000P =16      // MC8000Pデバイスドライバが同時に認識するボードの最大数(16枚)
}

```

（使用例）MC8000Pボードの最大数を指定する場合は `CONST.MAX_BOARD_MC8000P`

```

public enum MaxValue : uint
{
    MCX304          =268435455, // MCX304の出力パルスの最大値
    MCX314As        =4294967295 // MCX314Asの出力パルスの最大値
}

```

（使用例）MCX304の出力パルスの最大値を指定する場合は `MaxValue.MCX304`

⑦ I C 名

[C#]

```

public enum IC : int
{
    A           =0,      // 1つ目のIC
    B           =1,      // 2つ目のIC
    MCX304      =0,      // MCX304
    MCX314AS    =1,      // MCX314AS
}

```

（使用例）I C Aを指定する場合は `IC.A`

(2) 軸指定の方法は、下記の通りです。

軸	VC、VB.NET
X	AXIS_X
Y	AXIS_Y
Z	AXIS_Z
U	AXIS_U
全軸	AXIS_ALL

① 1 軸指定の場合

AXIS_X, AXIS_Y, AXIS_Z, AXIS_U のいずれかを指定します。

(使用例) X軸にドライブ速度 1000 を設定する

```
[VC]      Nmc_Speed(No, IcNo, AXIS_X, 1000);
[VB.NET]  Call Nmc_Speed(No, IcNo, AXIS_X, 1000)
[C#]      MC8000P.Nmc_Speed(No, IcNo, AXIS.X, 1000);
```

② 2 軸指定の場合

ビットOR演算子を使用します。

例えばX軸とY軸を一度に指定する場合、

```
[VC] . . . . AXIS_X | AXIS_Y を指定します。
[VB.NET] . . . . AXIS_X Or AXIS_Y を指定します。
[C#] . . . . AXIS.X | AXIS.Y を指定します。
```

(使用例) X, Y軸にドライブ速度 1000 を設定する

```
[VC]      Nmc_Speed(No, IcNo, AXIS_X | AXIS_Y, 1000);
[VB.NET]  Call Nmc_Speed(No, IcNo, AXIS_X Or AXIS_Y, 1000)
[C#]      MC8000P.Nmc_Speed(No, IcNo, AXIS.X | AXIS.Y, 1000);
```

③ 3 軸指定の場合

ビットOR演算子を使用します。

例えばX軸とY軸とZ軸を一度に指定する場合、

```
[VC] . . . . AXIS_X | AXIS_Y | AXIS_Z を指定します。
[VB.NET] . . . . AXIS_X Or AXIS_Y Or AXIS_Z を指定します。
[C#] . . . . AXIS.X | AXIS.Y | AXIS.Z を指定します。
```

(使用例) X, Y, Z軸にドライブ速度 1000 を設定する

```
[VC]      Nmc_Speed(No, IcNo, AXIS_X | AXIS_Y | AXIS_Z, 1000);
[VB.NET]  Call Nmc_Speed(No, IcNo, AXIS_X Or AXIS_Y Or AXIS_Z, 1000)
[C#]      MC8000P.Nmc_Speed(No, IcNo, AXIS.X | AXIS.Y | AXIS.Z, 1000);
```

④ 全軸指定の場合

AXIS_ALL を指定します。

(使用例) 全軸にドライブ速度 1000 を設定する

```
[VC]      Nmc_Speed(No, IcNo, AXIS_ALL, 1000);
[VB.NET]  Call Nmc_Speed(No, IcNo, AXIS_ALL, 1000)
[C#]      MC8000P.Nmc_Speed(No, IcNo, AXIS.ALL, 1000);
```

(3) 補間関数で使用する構造体 (VB.NETではユーザー定義型) の定義は、下記の通りです。 ※MC8043P/MC8043Peのみ

① V C

```
// 2 軸 B P 補間
typedef struct _DATA_2BP
{
    USHORT Bp1p;      // BP1Pデータ
    USHORT Bp1m;      // BP1Mデータ
    USHORT Bp2p;      // BP2Pデータ
    USHORT Bp2m;      // BP2Mデータ
} DATA_2BP;

// 3 軸 B P 補間
typedef struct _DATA_3BP
{
    USHORT Bp1p;      // BP1Pデータ
    USHORT Bp1m;      // BP1Mデータ
    USHORT Bp2p;      // BP2Pデータ
    USHORT Bp2m;      // BP2Mデータ
    USHORT Bp3p;      // BP3Pデータ
    USHORT Bp3m;      // BP3Mデータ
} DATA_3BP;
```

```

// 2軸連続補間
typedef struct _DATA_2CIP
{
    USHORT Command;    // 命令番号 (CMD_IP_2ST, CMD_IP_CW, CMD_IP_CCWのいずれかをセットする)
    USHORT Speed;      // 速度 (速度を変更する場合は1～8000、変更しない場合は0をセットする)
    long EndP1;        // 終点 (第1軸)
    long EndP2;        // 終点 (第2軸)
    long Center1;      // 円弧中心点 (第1軸)
    long Center2;      // 円弧中心点 (第2軸)
} DATA_2CIP;        // 注: 第1軸、第2軸は、WR5で指定する

// 3軸連続補間
typedef struct _DATA_3CIP
{
    long EndP1;        // 終点 (第1軸)
    long EndP2;        // 終点 (第2軸)
    long EndP3;        // 終点 (第3軸)
    USHORT Speed;      // 速度 (速度を変更する場合は1～8000、変更しない場合は0をセットする)
} DATA_3CIP;        // 注: 第1軸、第2軸、第3軸は、WR5で指定する

```

② V B . N E T

' 2軸BP補間

```

Structure DATA_2BP
    Dim Bp1p As Short    ' BP1Pデータ
    Dim Bp1m As Short    ' BP1Mデータ
    Dim Bp2p As Short    ' BP2Pデータ
    Dim Bp2m As Short    ' BP2Mデータ
End Structure

```

' 3軸BP補間

```

Structure DATA_3BP
    Dim Bp1p As Short    ' BP1Pデータ
    Dim Bp1m As Short    ' BP1Mデータ
    Dim Bp2p As Short    ' BP2Pデータ
    Dim Bp2m As Short    ' BP2Mデータ
    Dim Bp3p As Short    ' BP3Pデータ
    Dim Bp3m As Short    ' BP3Mデータ
End Structure

```

' 2軸連続補間

```

Structure DATA_2CIP
    Dim Cmd As Short    ' 命令番号 (CMD_IP_2ST, CMD_IP_CW, CMD_IP_CCWのいずれかをセットする)
    Dim Speed As Short  ' 速度 (速度を変更する場合は1～8000、変更しない場合は0をセットする)
    Dim EndP1 As Integer ' 終点 (第1軸)
    Dim EndP2 As Integer ' 終点 (第2軸)
    Dim Center1 As Integer ' 円弧中心点 (第1軸)
    Dim Center2 As Integer ' 円弧中心点 (第2軸)
End Structure        ' 注: 第1軸、第2軸は、WR5で指定する

```

' 3軸連続補間

```

Structure DATA_3CIP
    Dim EndP1 As Integer ' 終点 (第1軸)
    Dim EndP2 As Integer ' 終点 (第2軸)
    Dim EndP3 As Integer ' 終点 (第3軸)
    Dim Speed As Short   ' 速度 (速度を変更する場合は1～8000、変更しない場合は0をセットする)
End Structure        ' 注: 第1軸、第2軸、第3軸は、WR5で指定する

```

③C #

```

// 2軸BP補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_2BP
{
    public DATA_2BP(ushort bp1p, ushort bp1m, ushort bp2p, ushort bp2m)
    {
        this.Bp1p    = bp1p;
        this.Bp1m    = bp1m;
        this.Bp2p    = bp2p;
        this.Bp2m    = bp2m;
    }
    public ushort Bp1p; // BP1Pデータ
    public ushort Bp1m; // BP1Mデータ
    public ushort Bp2p; // BP2Pデータ
    public ushort Bp2m; // BP2Mデータ
}

// 3軸BP補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_3BP
{
    public DATA_3BP(ushort bp1p, ushort bp1m, ushort bp2p, ushort bp2m, ushort bp3p, ushort bp3m)
    {
        this.Bp1p    = bp1p;
        this.Bp1m    = bp1m;
        this.Bp2p    = bp2p;
        this.Bp2m    = bp2m;
        this.Bp3p    = bp3p;
        this.Bp3m    = bp3m;
    }
    public ushort Bp1p; // BP1Pデータ
    public ushort Bp1m; // BP1Mデータ
    public ushort Bp2p; // BP2Pデータ
    public ushort Bp2m; // BP2Mデータ
    public ushort Bp3p; // BP3Pデータ
    public ushort Bp3m; // BP3Mデータ
}

// 2軸連続補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_2CIP
{
    public DATA_2CIP(ushort command, ushort speed, int endp1, int endp2, int center1, int center2)
    {
        this.Command = command;
        this.Speed   = speed;
        this.EndP1   = endp1;
        this.EndP2   = endp2;
        this.Center1 = center1;
        this.Center2 = center2;
    }
    public ushort Command; // 命令番号 (CMD_IP_2ST, CMD_IP_CW, CMD_IP_CCWのいずれかをセットする)
    public ushort Speed; // 速度 (指定する場合は1~8000、指定しない場合は0をセットする)
    public int EndP1; // 終点 (第1軸)
    public int EndP2; // 終点 (第2軸)
    public int Center1; // 円弧中心点 (第1軸)
    public int Center2; // 円弧中心点 (第2軸)
}
// 注: 第1軸、第2軸は、WR5で指定する

```

```
// 3軸連続補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_3CIP
{
    public DATA_3CIP(int endp1, int endp2, int endp3, ushort speed)
    {
        this.EndP1 = endp1;
        this.EndP2 = endp2;
        this.EndP3 = endp3;
        this.Speed = speed;
    }
    public int EndP1; // 終点 (第1軸)
    public int EndP2; // 終点 (第2軸)
    public int EndP3; // 終点 (第3軸)
    public ushort Speed; // 速度 (指定する場合は1~8000、指定しない場合は0をセットする)
} // 注: 第1軸、第2軸、第3軸は、WR5で指定する
```

構造体の使用例を以下に示します。

例1. 定義、初期化が別の場合

```
DATA_2BP [] Data2Bp = new DATA_2BP[4]; // 2軸BP補間データ

// 補間データ設定
Data2Bp[0].Bp1p = 0xFF30; // 1111 1111 0011 0000 BP1+方向 10パルス
Data2Bp[0].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data2Bp[0].Bp2p = 0; // 0000 0000 0000 0000 BP2+方向 0パルス
Data2Bp[0].Bp2m = 0x84FF; // 1000 0100 1111 1111 BP2-方向 10パルス

Data2Bp[1].Bp1p = 0xAC35; // 1010 1100 0011 0101 BP1+方向 8パルス
Data2Bp[1].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data2Bp[1].Bp2p = 0xC000; // 1100 0000 0000 0000 BP2+方向 2パルス
Data2Bp[1].Bp2m = 0x36E7; // 0011 0110 1110 0111 BP2-方向 10パルス

Data2Bp[2].Bp1p = 0x3F3F; // 0011 1111 0011 1111 BP1+方向 12パルス
Data2Bp[2].Bp1m = 0xC000; // 1100 0000 0000 0000 BP1-方向 2パルス
Data2Bp[2].Bp2p = 0xFBDA; // 1111 1011 1101 1010 BP2+方向 12パルス
Data2Bp[2].Bp2m = 0; // 0000 0000 0000 0000 BP2-方向 0パルス

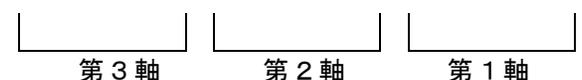
Data2Bp[3].Bp1p = 0; // 0000 0000 0000 0000 BP1+方向 0パルス
Data2Bp[3].Bp1m = 0x1CF2; // 0001 1100 1111 0010 BP1-方向 8パルス
Data2Bp[3].Bp2p = 0xFFFF; // 1111 1111 1111 1111 BP2+方向 16パルス
Data2Bp[3].Bp2m = 0; // 0000 0000 0000 0000 BP2-方向 0パルス
```

例2. 定義と初期化を同時に行う場合

```
// 2軸BP補間データ BP1P, BP1M, BP2P, BP2M (MCX314As取説 図2.32のデータ)
DATA_2BP[] Data2Bp = new DATA_2BP[]
{
    new DATA_2BP(0x0000, 0x2BFF, 0xFFD4, 0x0000),
    new DATA_2BP(0xF6FE, 0x0000, 0x000F, 0x3FC0),
    new DATA_2BP(0x1FDB, 0x0000, 0x00FF, 0xFC00),
    new DATA_2BP(0x4000, 0x7FF5, 0x0000, 0x0AFF),
};
```

- (4) 補間関数で指定する補間軸(IpAxis)の指定は下記の通りです。 ※MC8043P/MC8043Peのみ
4軸X、Y、Z、Uの中から補間する軸の組み合わせを、次の16ビットデータの低位6ビットに設定します。2軸補間の場合は第1軸、第2軸に、第3軸補間の場合は第1軸、第2軸、第3軸に設定します。

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	AX31	AX30	AX21	AX20	AX11	AX10



◆各ビットの説明

D1, 0 AX11, 10 補間ドライブを行う第1軸(主軸)を指定します。軸コードを下表に示します。

軸	コード(2進)
X	0 0
Y	0 1
Z	1 0
U	1 1

第1軸：X、第2軸：Y、第3軸：Zの例

D5 D4 D3 D2 D1 D0
1 0 0 1 0 0

D3, 2 AX21, 20 補間ドライブを行う第2軸を上表に示すコードで指定します。

D5, 4 AX31, 30 3軸補間ドライブを行う第3軸を上表に示すコードで指定します。
2軸補間ドライブの時は使用しないので、何をセットしても構いません。

◆C#の場合、次の既定値を使い補間軸を指定して下さい。

```
public enum IP_AXIS : int
{
    IP_X = 0, // 補間軸X
    IP_Y = 1, // 補間軸Y
    IP_Z = 2, // 補間軸Z
    IP_U = 3, // 補間軸U
}
```

軸	使用例
X	IP_AXIS.IP_X
Y	IP_AXIS.IP_Y
Z	IP_AXIS.IP_Z
U	IP_AXIS.IP_U

補間軸の指定方法は、次の例のように、ビット演算を用いて設定します。

(使用例) 第1軸：X、第2軸：Y、第3軸：Zの3軸補間の場合、次のようにビット演算をして設定してください。

IpAxis = IP_AXIS.IP_X | IP_AXIS.IP_Y<<2 | IP_AXIS.IP_Z<<4 ;

(5) 連続補間関数実行時の速度変更について ※MC8043P/MC8043Peのみ

連続補間関数は、補間実行中に速度変更を行う事ができます。各セグメント毎に速度を設定できます。補間実行中に速度変更を行う場合は、関数のパラメータSpdChgFlgにTRUE(True)を設定します。

◆各セグメント毎の速度の設定方法

DATA_2CIPのSpeed、あるいはDATA_3CIPのSpeed に、各セグメントの速度を設定します。

- ・前のセグメントと異なる速度にする場合は、1~8000 の速度を設定します。
- ・前のセグメントと同じ速度のままにする場合は、0 を設定します。

◆速度を変更するタイミングについて

DDATA_2CIPのSpeed、あるいはDATA_3CIPのSpeed設定値が 1~8000 の場合の補間関数の処理内容について説明します。

1つ目のセグメントについては、セグメント実行前に速度を設定します。

2つ目以降のセグメントについては、該当セグメントが開始した直後(その次のセグメントが書き込み可能になった時)に速度を設定します。

例えば、2つ目が開始し3つ目のセグメントが書き込み可能になった時に、2つ目のセグメントの速度を設定します。この為、2つ目が開始しても2つ目の速度を設定するまでの間は1つ目の速度で実行されますので、ご注意下さい。

(6) アドレスの指定

Nmc_OutPort, Nmc_InPort関数のアドレスには、各ボードの取扱説明書に記載している I/Oアドレスを指定します。

ライトレジスタ、リードレジスタなどの I/Oアドレスは下記の通りです。詳細は各ボードの取扱説明書を参照。

ボード名	ライトレジスタのアドレス	リードレジスタのアドレス	PIX132のアドレス
MC8082P / MC8082Pe	0 ~ F	0 ~ F	14 ~ 16
MC8043P / MC8043Pe	0 ~ 7	0 ~ 7	-----

注1：アドレスは16進数で記載しています。

注2：ライトレジスタ、リードレジスタにアクセスする場合、Nmc_OutPort, Nmc_InPort関数ではなく、専用の関数を使用した方が便利です。

(7) IC番号の指定は下記の通りです。

① ICを1つ搭載しているボードの場合は0を指定します。

② ICを2つ搭載しているボードの場合は次の値を指定します。

- ・ IC-Aの場合は0
- ・ IC-Bの場合は1

各ボードの搭載IC数と指定するIC番号は次の通りです。

ボード名	搭載IC数	IC番号
MC8082P / MC8082Pe	2	0, 1
MC8043P / MC8043Pe	1	0

注：ICとはMCX314AsまたはMCX304の事を指します。

4.1.4 使用方法

■API関数宣言

API関数宣言は、下記の場所で行っています。

VC++	: MC8000P_DLL.H
VB.NET	: MC8000P_DLL.vb
C#	: 取扱説明書または入力支援を利用してください。

■使用方法

- (1) 開始処理・・・各関数を使用する前にNmc_Openを一度実行して下さい。
- (2) 終了処理・・・プログラム終了時にNmc_Close、又はNmc_CloseAllを実行して下さい。

■ボード番号について

アプリケーションから指定するボード番号は下記の通りです。

	ボードのロータリスイッチの値	アプリケーションから指定するボード番号(10進数)
1	0 ~ 9	0 ~ 9
2	A ~ F	10 ~ 15

■関数使用時の注意

- (1) VC, VB.NET, C# (全ての言語) について

- ①Nmc_Open関数実行前に各関数を実行した場合の動作保証はできません。
- ②接続していないボードの番号を誤って指定した場合も、各関数の動作の保証はできません。
- ③1枚のボードに対して、2つ以上のアプリケーションから同時にアクセス(オープンなど)を行わないで下さい。

- (2) VC, C#のみ

- ①割り込み処理関数を使用する場合はWindowsの性格上、割り込み発生からユーザー定義ルーチンへ制御が移行するまでの時間を保証することは出来ません。
- ②割り込みを行う場合は、割り込みユーザー関数(Nmc_SetEventで指定した関数)を実行中にクローズ処理(Nmc_Close又はNmc_CloseAll)を実行しないで下さい。
クローズ処理を行う場合は、必ず、割り込みユーザー関数が終了している状態で行って下さい。

■VCにて割り込みを処理する方法

- ①Nmc_Open関数にて割り込みを使用する設定にします。

```
Nmc_Open(No, TRUE); // 第2引数 TRUE : 割り込みを使用する FALSE : 割り込みを使用しない
```

- ②Nmc_SetEvent関数にて割り込みを処理するユーザー関数を設定します。また、許可する割り込みを設定します。

```
Nmc_SetEvent(No, MC_EventProc, lpParam); // ユーザー関数のアドレスと引数を設定  
Nmc_WriteReg1(No, IcNo, AXIS_ALL, 0x8000); // 指定したICの停止時割り込み発生(全軸)
```

- ③割り込みが発生すると、Nmc_SetEvent関数で設定した割り込みユーザー関数が呼び出されます。
割り込みユーザー関数では、割り込み要因を確認します。RR3の割り込み要因は、Nmc_ReadEvent関数にて取得します。

■割り込みユーザー関数例

```
DWORD WINAPI MC_EventProc(LPVOID lpParam)  
{  
    . . . . .  
    long Rr3X, Rr3Y, Rr3Z, Rr3U;  
    Nmc_ReadEvent(No, IcNo, &Rr3X, &Rr3Y, &Rr3Z, &Rr3U);  
    . . . . .  
    return 0;  
}
```

- ④割り込み処理するユーザー関数の設定を解除する場合は Nmc_ResetEvent を実行して下さい。
この関数を実行すると、ボードで割り込みが発生してもユーザー関数は呼び出されません。
Nmc_ResetEvent(No);

■ C#にて割り込みを処理する方法

- ①MC8000P.Nmc_SetEventにて、割り込みを処理するメソッドを設定します。
複数枚ボードを使用する場合は、下記のように設定します。

(ボード番号0の場合)

```
MC8000P.callback[0] = new MC8000P.UserThread(isr);           // isrは割り込みユーザー関数
bool ret = MC8000P.Nmc_SetEvent(0, MC8000P.callback[0], param); // 第1引数にボード番号0を指定
                                                                // 関数のアドレスと引数を設定
MC8000P.Nmc_WriteReg1(0, (int)IC.A, AXIS.ALL, 0x8000);      // 停止時割り込み発生(全軸)
. . .
```

(ボード番号1の場合)

```
MC8000P.callback[1] = new MC8000P.UserThread(isr2);        // isrは割り込みユーザー関数
bool ret = MC8000P.Nmc_SetEvent(1, MC8000P.callback[1], param); // 第1引数にボード番号1を指定
                                                                // 関数のアドレスと引数を設定
MC8000P.Nmc_WriteReg1(1, (int)IC.A, AXIS.ALL, 0x8000);      // 停止時割り込み発生(全軸)
. . .
```

- ②割り込み処理をする関数では、各ボードの割り込み要因を確認します。RR3の割り込み要因はMC8000P.Nmc_ReadEventにて取得します。

(ボード番号0の割り込みユーザー関数)

```
static void isr(int param)
{
    int Rr3X, Rr3Y, Rr3Z, Rr3U;
    MC8000P.Nmc_ReadEvent(0, (int)IC.A, out Rr3X, out Rr3Y, out Rr3Z, out Rr3U);
    . . . . .
}
```

(ボード番号1の割り込みユーザー関数)

```
static void isr2(int param)
{
    int Rr3X, Rr3Y, Rr3Z, Rr3U;
    MC8000P.Nmc_ReadEvent(1, (int)IC.A, out Rr3X, out Rr3Y, out Rr3Z, out Rr3U);
    . . . . .
}
```

- ③割り込み処理をする関数の設定を解除する場合は、MC8000P.Nmc_ResetEventメソッドを使用します。
このメソッドを実行すると、ボードで割り込みが発生しても割り込みユーザー関数のメソッドは呼び出されません。

■連続補間について ※MC8043P/MC8043Peのみ

連続補間を行う場合、MCX314As取扱説明書の「2.4.5 連続補間」を必ず参照し、その章に記載している処理をアプリケーションで行ってください。また、連続補間関数(注1)ではこれらの一部の処理をDLLで行っていますので、この連続補間関数を使用して連続補間の処理を行うこともできます。但し、連続補間関数を使用する場合はいくつか注意事項がありますので、ご注意ください。

注1 : Nmc_2CIPExec, Nmc_3CIPExec, Nmc_2CIPExec_BG, Nmc_3CIPExec_BG

連続補間関数を使用する場合の注意事項 :

連続補間関数は、次のセグメントの終点や中心点などを書き込み、補間命令書き込み後、エラーチェックを行い、エラー発生の場合は関数を終了します。エラーが発生していない場合は、次のセグメントデータ書き込み可能チェック(RR0のD9ビットチェック)を行い、可能になった場合は次のセグメントデータや補間命令を書き込みます。本関数は、連続補間が終了するまでこの処理を繰り返します。エラーチェックと次のセグメントデータ書き込み可能チェック処理などがDLL内部で常にループしているため、本関数実行中にアプリケーションで他の処理も行いたい場合は、本関数の使用が適さない場合があります。この場合は、連続補間関数は使用せず、MCX314As取扱説明書を参考にしてアプリケーションで連続補間の処理を作成してください。連続補間の加減速ドライブについてはMCX314As取扱説明書の「2.4.6 加減速ドライブでの補間」を参照してください。また、連続補間関数を使用する場合は、関数を実行する前に初速度を8000に設定してください。(本関数実行中に初速度を変更しないでください。)この場合、各セグメント内は定速ドライブになります。

■ B P 補間（ビットパターン補間）について

※MC8043P/MC8043Peのみ

B P 補間を行う場合、MCX314As取扱説明書の「2.4.3 ビットパターン補間」を必ず参照し、その章に記載している処理をアプリケーションで行ってください。また、B P 補間関数（注2）ではこれらの一部の処理をDLLで行っていますので、このB P 補間関数を使用してB P 補間の処理を行うこともできます。但し、B P 補間関数を使用する場合はいくつか注意事項がありますので、ご注意ください。

注2 : Nmc_2BPExec, Nmc_3BPExec, Nmc_2BPExec_BG, Nmc_3BPExec_BG

B P 補間関数を使用する場合の注意事項：

B P 補間関数は、次のB P データを書き込み、補間命令書き込み後、エラーチェックを行い、エラー発生の場合は関数を終了します。エラーが発生していない場合は、スタックカウンタが2以下になったかチェック（RR0のD14, 13ビットチェック）を行い、2以下になった場合は次のB P データを書き込みます。本関数は、B P 補間が終了するまでこの処理を繰り返します。エラーチェックとスタックカウンタチェック処理などがDLL内部で常にループしているため、本関数実行中にアプリケーションで他の処理も行いたい場合は、本関数の使用が適さない場合があります。この場合は、B P 補間関数は使用せず、MCX314As取扱説明書を参考にしてアプリケーションでB P 補間の処理を作成してください。また、B P 補間の加減速ドライブについてはMCX314As取扱説明書の「2.4.6 加減速ドライブでの補間」を参照してください。

■ 補間関数使用時の注意

※MC8043P/MC8043Peのみ

- (1) 下記の補間関数について、一度に実行できるのは1つの補間関数のみです。
補間関数実行中に、別の補間関数は実行できません。実行した場合、エラーが返ります。

Nmc_2BPExec	Nmc_2BPExec_BG	Nmc_2CIPExec	Nmc_2CIPExec_BG
Nmc_3BPExec	Nmc_3BPExec_BG	Nmc_3CIPExec	Nmc_3CIPExec_BG

- (2) 上記の補間関数実行中は、下記の処理を実行しないで下さい。

- ①補間命令（30h～3Dh）の実行
- ②WR 5 補間モードレジスタの変更

- (3) 下記の補間関数はバックグラウンドで実行する為、補間関数開始時に補間データ用のメモリを確保し、ユーザーが指定した補間データをコピーします。そして、バックグラウンドで実行していた補間処理が終了する時にそのメモリを解放し、ユーザーウィンドウにメッセージを送信します。
この為、下記補間関数がバックグラウンドで実行している最中にクローズ処理（Nmc_Close 又は Nmc_CloseAll）を実行しないで下さい。
また、下記補間関数がバックグラウンドで実行している最中にアプリケーションを終了しないで下さい。
補間関数の実行を途中で止めたい時は、補間中断関数(Nmc_IPStop)を実行し、必ず中断メッセージを受け取って下さい。

Nmc_2BPExec_BG	Nmc_2CIPExec_BG
Nmc_3BPExec_BG	Nmc_3CIPExec_BG

- (4) 補間関数にて補間実行中、速度が速い場合は次のデータセットが間に合わず、補間が停止する場合があります。

■マルチスレッドアプリケーション開発時の注意

ここでは、マルチスレッドで動作するアプリケーションを開発する際の注意事項を説明します。

Nmc_xxx の関数では、軸切り替え処理、WR 6，WR 7 にデータを書き込む処理、RR 6，RR 7 にデータを読み出す処理を実行している関数があります。それぞれの Nmc_xxx 関数は次の通りです。

◆軸切り替え処理を実行している関数

Nmc_Reset	Nmc_Command	Nmc_Command_IP			
Nmc_WriteReg0	Nmc_WriteReg1	Nmc_WriteReg2	Nmc_WriteReg3		
Nmc_ReadReg1	Nmc_ReadReg2				
Nmc_Range	Nmc_Jerk	Nmc_Acc	Nmc_Dec	Nmc_StartSpd	Nmc_Speed
Nmc_Pulse	Nmc_Pulse_VB	Nmc_DecP	Nmc_DecP_VB	Nmc_Center	Nmc_Lp
Nmc_Ep	Nmc_CompP	Nmc_CompM	Nmc_AccOfst	Nmc_DJerk	Nmc_HomeSpd
Nmc_ExpMode	Nmc_SyncMode	Nmc_HomeMode			
Nmc_ReadLp	Nmc_ReadEp	Nmc_ReadSpeed	Nmc_ReadAccDec	Nmc_ReadSyncBuff	
Nmc_2BPExec	Nmc_3BPExec	Nmc_2BPExec_BG	Nmc_3BPExec_BG		
Nmc_2CIPExec	Nmc_3CIPExec	Nmc_2CIPExec_BG	Nmc_3CIPExec_BG		
Nmc_WriteRegSetAxis	Nmc_ReadRegSetAxis	Nmc_WriteData	Nmc_WriteData2	Nmc_ReadData	

◆WR 6，WR 7 にデータを書き込む処理を実行している関数

Nmc_Range	Nmc_Jerk	Nmc_Acc	Nmc_Dec	Nmc_StartSpd	Nmc_Speed
Nmc_Pulse	Nmc_Pulse_VB	Nmc_DecP	Nmc_DecP_VB	Nmc_Center	Nmc_Lp
Nmc_Ep	Nmc_CompP	Nmc_CompM	Nmc_AccOfst	Nmc_DJerk	Nmc_HomeSpd
Nmc_ExpMode	Nmc_SyncMode	Nmc_HomeMode	Nmc_WriteData	Nmc_WriteData2	
Nmc_2CIPExec	Nmc_3CIPExec	Nmc_2CIPExec_BG	Nmc_3CIPExec_BG		
Nmc_WriteReg6	Nmc_WriteReg7				
Nmc_OutPortまたはNmc_WriteRegでWR6, WR7に書いた場合					

◆RR 6，RR 7 にデータを読み出す処理を実行している関数

Nmc_ReadLp	Nmc_ReadEp	Nmc_ReadSpeed	Nmc_ReadAccDec	Nmc_ReadSyncBuff
Nmc_ReadData				

WR 1～WR 3 書き込み、RR 1～RR 2 読み出し、データ書き込み命令、データ読み出し命令を実行する場合、基本的には次の Nmc_xxx 関数を使用して下さい。

◆WR 1～WR 3 書き込み

Nmc_WriteReg1	Nmc_WriteReg2	Nmc_WriteReg3	Nmc_WriteRegSetAxis
---------------	---------------	---------------	---------------------

◆RR 1～RR 2 読み出し

Nmc_ReadReg1	Nmc_ReadReg2	Nmc_ReadRegSetAxis
--------------	--------------	--------------------

◆データ書き込み命令

Nmc_Range	Nmc_Jerk	Nmc_Acc	Nmc_Dec	Nmc_StartSpd	Nmc_Speed
Nmc_Pulse	Nmc_Pulse_VB	Nmc_DecP	Nmc_DecP_VB	Nmc_Center	Nmc_Lp
Nmc_Ep	Nmc_CompP	Nmc_CompM	Nmc_AccOfst	Nmc_DJerk	Nmc_HomeSpd
Nmc_ExpMode	Nmc_SyncMode	Nmc_HomeMode	Nmc_WriteData	Nmc_WriteData2	

◆データ読み出し命令

Nmc_ReadLp	Nmc_ReadEp	Nmc_ReadSpeed	Nmc_ReadAccDec	Nmc_ReadSyncBuff
Nmc_ReadData				

WR 1～WR 3 書き込み、RR 1～RR 2 読み出し、データ書き込み命令、データ読み出し命令を実行する際、これらの関数を使用せずに同じ処理を行った場合、マルチスレッド環境では注意が必要です。

(1) 例えば、WR 1 書き込み時には Nmc_WriteReg1 を使用しますが、それ以外の方法としては下記の方法などがあります。

- ① Nmc_OutPort(No, MCX_WR0, 0x010F); // X軸に切り替える (IC-A)
- ② Nmc_OutPort(No, MCX_WR1, Data); // WR 1 書き込み (IC-A)

また、次のような関数でも同じ処理を実行できます。

- ③ Nmc_WriteReg(No, IcNo, MCX_WR0, 0x010F); // X軸に切り替える
- ④ Nmc_WriteReg(No, IcNo, MCX_WR1, Data); // WR 1 書き込み

この場合、①と②の間、③と④の間に、軸を切り替える Nmc_xxx 関数が実行された場合、別の軸のWR 1 にデータが書かれてしまいます。

(2) 例えば、速度設定時には Nmc_Speed を使用しますが、それ以外の方法としては下記の方法などがあります。

- ① Nmc_OutPort(No, MCX_WR6, Data); // WR 6 書き込み (IC-A)
- ② Nmc_OutPort(No, MCX_WR0, 0x0105); // WR 6 データを X 軸の速度に設定する (IC-A)

また、次のような関数でも同じ処理を実行できます。

- ③ Nmc_WriteReg6(No, IcNo, Data); // WR 6 書き込み
- ④ Nmc_Command(No, IcNo, AXIS_X, 0x05); // WR 6 データを X 軸の速度に設定する

この場合、①と②の間、③と④の間に、WR 6, WR 7 にデータを書き込む Nmc_xxx 関数が実行された場合、別のデータが速度に書かれてしまいます。

(3) 例えば、論理位置カウンタ読み出し時には Nmc_ReadLp を使用しますが、それ以外の方法としては下記の方法などがあります。

- ① Nmc_OutPort(No, MCX_WR0, 0x0110); // X 軸の論理位置カウンタを RR 6, RR 7 に読み出す (IC-A)
- ② d6 = Nmc_InPort(No, MCX_RR6); // RR 6 を読み出す (IC-A)
- ③ d7 = Nmc_InPort(No, MCX_RR7); // RR 7 を読み出す (IC-A)

この場合、①と②の間、②と③の間に、RR 6, RR 7 にデータを読み出す Nmc_xxx 関数が実行された場合、ここでは別のデータが読み出されてしまいます。

このように、マルチスレッド環境では、目的の処理を実行するのに 2 回以上 API 関数をコールする場合は、そのような処理を行わないようにするか、あるいは、排他制御を行う必要があります。

Nmc_xxx の関数を 1 回コールして処理が完結する場合は、マルチスレッド環境でも問題なく動作します。
Nmc_xxx の各関数同士は排他制御を行っています。

4.2 プログラミング上の注意点

(1) 入力信号フィルタの初期設定

■MC8043P/MC8043Peの場合(MCX314As搭載ボード)

ボードのリミット信号などの各入力信号は、MCX314As内蔵の積分フィルタを使用します。ノヴァエレクトロニクスより供給されるデバイスドライバでは、パソコン起動時の初期設定において、MCX314Asに拡張モード設定コマンド(60h)を発行して各入力信号に対して、以下のようにフィルタを設定しています。

フィルタ遅延時間：512 μ sec

各信号のフィルタの有効/無効：

信号名	有効 / 無効
EMG, nLMT+, nLMT-, nIN0, nIN1	有効
nIN2	有効
nINPOS, nALARM	有効
nEXOP+, nEXOP-	有効
nIN3	有効

アプリケーション上で、これらの入力信号フィルタの有効/無効を変更する場合には、MCX314Asの取扱説明書6.16節を参照してください。拡張モード設定コマンド(60h)によって変更することが出来ます。次の記述例では、ボード番号0のX, Y, Z, U全軸に対して、上記の初期設定と同じ内容を設定しています。Nmc_ExpModeは拡張モード設定コマンド(60h)を実行しています。

(例1)

```
Nmc_ExpMode(0, 0, AXIS_ALL, 0x5F00, 0x0000);
```

(例2)

```
Nmc_WriteReg6(0, 0, 0x5F00);
```

```
Nmc_WriteReg7(0, 0, 0x0000);
```

```
Nmc_WriteReg0(0, 0, 0x0F60);
```

注意：

①拡張モード設定コマンド(60h)は、入力信号フィルタの設定(WR6)とともに、自動原点出しの設定(WR7)も行ないます。一方の設定を行なう場合においても必ず両方(WR6, 7)に適正值を設定してください。

②アプリケーション上からソフトリセット(WR0/D15に1セット)した場合にも、各入力信号のフィルタは上記のパソコン起動時初期設定と同じ設定がデバイスドライバ内で行なわれます。

■MC8082P/MC8082Pe、MC8022P、MC8042Pの場合 (MCX304搭載ボード)

ボードのリミット信号などの各入力信号は、MCX304内蔵の積分フィルタを使用します。ノヴァエレクトロニクスより供給されるデバイスドライバでは、パソコン起動時の初期設定において、MCX304のモードレジスタ 3 (WR3) にデータを書き込み、各入力信号に対して以下のようにフィルタを設定しています。

フィルタ遅延時間 : 512 μ sec

各信号のフィルタの有効/無効 :

信号名	有効 / 無効
EMG, nLMT+, nLMT-, nSTOP0, nSTOP1	有効
nSTOP2	有効
nINPOS, nALARM	有効
nEXOP+, nEXOP-	有効

アプリケーション上で、これらの入力信号フィルタの有効/無効を変更する場合には、MCX304の取扱説明書2.6.9節、4.6節を参照してください。モードレジスタ 3 (WR3) にデータを書き込む事によって変更することが出来ます。次の記述例では、ボード番号 0 の全 I C の X, Y, Z, U 全軸に対して、上記の初期設定と同じ内容を設定しています。

(例)

```
Nmc_WriteReg3(0, 0, AXIS_ALL, 0x4F00); // I C - A の設定
Nmc_WriteReg3(0, 1, AXIS_ALL, 0x4F00); // I C - B の設定 (複数 I C の場合)
```

注意 :

アプリケーション上からソフトリセット (WR0/D15に 1 セット) した場合にも、各入力信号のフィルタは上記のパソコン起動時初期設定と同じ設定がデバイスドライバ内で行なわれます。

(2) パソコンのスタンバイ、休止動作

本デバイスドライバは、パソコンのスタンバイ動作、あるいは休止動作を行った後のドライバの動作を保証していません。パソコンのスタンバイ動作、あるいは休止動作を行った後に、ボードにアクセスしたい場合は、一度パソコンを再起動させてから行って下さい。

(3) 割り込みサポートについて

割り込みは、VC++およびC#で開発したアプリケーションでのみサポートしています。VBで開発したアプリケーションでは割り込みをサポートしていません。

本デバイスドライバでサポートしている割り込みの種類は下記の通りです。

■MC8043P/MC8043Pe (MCX304搭載ボード)

- ・ R R 3 レジスタで報告される割り込み全て

■MC8082P/MC8082Pe、MC8022P、MC8042P (MCX314As搭載ボード)

- ・ R R 3 レジスタで報告される割り込み全て
- ・ 連続補間ドライブで次セグメントのデータと補間ドライブ命令が書き込み可能となった時の割り込み
- ・ ビットパターン補間において、スタックカウンタの値が 2 から 1 に変わった時の割り込み

(4) 割り込みクリアについて

①RR3レジスタで報告される割り込みについて

ボードでこの割り込みが発生した直後、ドライバ内でRR3を読み出し、割り込みをクリアしています。
その後、アプリケーションの割り込み用ユーザー関数が呼び出されます。(ユーザー関数を設定した場合のみ)

②連続補間ドライブで次セグメントのデータと補間ドライブ命令が書き込み可能となった時の割り込みについて (MC8043P/MC8043Peのみ)

ボードでこの割り込みが発生した直後、ドライバ内で補間割り込みをクリアしています。
その後、アプリケーションの割り込み用ユーザー関数が呼び出されます。(ユーザー関数を設定した場合のみ)

③ビットパターン補間において、スタックカウンタの値が2から1に変わった時の割り込みについて (MC8043P/MC8043Peのみ)

ボードでこの割り込みが発生した直後、ドライバ内で補間割り込みをクリアしています。
その後、アプリケーションの割り込み用ユーザー関数が呼び出されます。(ユーザー関数を設定した場合のみ)

(5) RR3割り込みと補間割り込みを両方使用する場合 (MC8043P/MC8043Peのみ)

RR3で報告される割り込みと補間割り込み(注1)を両方有効にする場合は、割り込みユーザー関数(注2)内で割り込要因を確認する際、RR3の割り込要因を先に読み出し、その後、補間割り込みが発生しているか確認してください。

例) 割り込み発生時の割り込みユーザー関数処理

- ①Nmc_ReadEventにてRR3の割り込要因を読み出し、RR3で割り込みが発生しているか確認する。
- ②補間割り込みが発生しているか確認する。(RR0のCNEXT、またはRR0のBPSC1,0を確認)

注1 : 連続補間ドライブで次セグメントのデータと補間ドライブ命令が書き込み可能となった時の割り込み、またはビットパターン補間において、スタックカウンタの値が2から1に変わった時の割り込み。

注2 : Nmc_SetEvent関数で指定したユーザー関数

4.3 MCX304評価ツール

MCX304 評価ツールプログラムは、MCX304を搭載しているボード(MC8082P/MC8082Pe、MC8022P、MC8042P)を評価するツールです。弊社ホームページよりダウンロードすることができます。(MC8000Pデバイスドライバソフトに添付) 評価ツールを実行する前に、MC8000Pデバイスドライバをインストールして下さい。

注：この章では、MCX304 評価ツールの概要について説明します。詳細については、Tool¥MCX304 Boardフォルダの ReadMe.txt(操作説明書)を参照して下さい。

4.3.1 実行プログラムについて

実行プログラムはMCX304-A.exeとMCX304-B.exeの2種類です。

■評価するMCX304について

各評価ツールが評価するMCX304は下記の通りです。

●MCX304-A.exe

- ①MCX304を1つ搭載しているボード(MC8022P、MC8042P)のMCX304
- ②MCX304を2つ搭載しているボード(MC8082P/MC8082Pe)の1つ目のMCX304 (MCX304-A)

●MCX304-B.exe

- ①MCX304を2つ搭載しているボード(MC8082P/MC8082Pe)の2つ目のMCX304 (MCX304-B)

■実行について

同じボードに対してMCX304-A.exeとMCX304-B.exeを同時に実行する事はできません。。異なるボードに対してMCX304-A.exe、またはMCX304-B.exeを同時に実行する事はできます。

4.3.2 機能概要

評価ツールを起動すると、ボード番号選択画面が表示され、ボード番号(ボードのロータリースイッチ番号(0~F))を選択することができます。ボード名表示ボタンを押すと、ボード番号の横にボード名(本ドライバを使用しているボードのみ)を表示することができます。ボード番号を選択後、OKボタンを押すと、メイン画面が表示されます。

メイン画面では、各軸パラメータ設定、ドライブ命令等の命令実行、現在位置・現在速度の表示、割り込み画面表示、パラメータ・モード設定値の保存、読み出し等を行います。また、モード設定画面やステータス画面を開き、モード設定、ステータス参照等を行う事ができます。Port A, B, C 出力画面でポートA, B, Cの汎用出力を行う事ができます。

4.3.3 メイン画面

メイン画面では、下図のような操作を行います。

ドライブ中の現在位置や
現在ドライブ速度等の表示

各軸のドライブ命令等の命令実行

位置カウンタ設定 →

各軸パラメータ設定

モード設定画面
ステータス画面
Port A, B, C出力画面の起動

各パラメータ、
モード設定値などの
保存、読み出し

X 軸ドライブ速度の軌跡表示
(プロット時のみ)

割り込み 閉じる

割り込み発生

このRR3は、Nmc_ReadEvent 関数で読み出しています。

	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
RR3	D-END C-STA C-END P≥C+ P<C+ P<C- P≥C-															
X	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○
Y	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Z	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
U	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

割り込み発生時に
割り込み画面表示

4.3.4 モード設定画面

モード設定画面では、MCX304のWR1～WR5（モードレジスタ、アウトプットレジスタ）を設定します。

モード設定																
	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
WR1	D-END	C-STA	C-END	P \geq C+	P<C+	P<C-	P \geq C-	SMOD	EPINV	EPCLR	SP2-E	SP2-L	SP1-E	SP1-L	SP0-E	SP0-L
X	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Z	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
U	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
WR2	INP-E	INP-L	ALM-E	ALM-L	PIND1	PIND0	PINMD	DIR-L	PLS-L	PLSMD	CMPSL	HLMT-	HLMT+	LMTMD	SLMT-	SLMT+
X	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Z	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
U	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
WR3	FL2	FL1	FL0	FE3	FE2	FE1	FE0	VRING	AVTRI	EXOP1	EXOP0	SACC	DSNDE	MANLD		
X	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Y	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Z	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
U	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
WR4	UOUT3	UOUT2	UOUT1	UOUT0	ZOUT3	ZOUT2	ZOUT1	ZOUT0	YOUT3	YOUT2	YOUT1	YOUT0	XOUT3	XOUT2	XOUT1	XOUT0
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
WR5	UOT3E	UOT2E	UOT1E	UOT0E	ZOT3E	ZOT2E	ZOT1E	ZOT0E	YOT3E	YOT2E	YOT1E	YOT0E	XOT3E	XOT2E	XOT1E	XOT0E
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

4.3.5 自動原点出しモード設定画面

自動原点出しモード設定画面では、MCX304の自動原点出しモードを設定します。

自動原点出しモード設定																
	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
WR6	DCCW2	DCCW1	DCCW0	DCC-L	DCC-E	LIMIT	SAND	PCLR	ST4-D	ST4-E	ST3-D	ST3-E	ST2-D	ST2-E	ST1-D	ST1-E
X	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>								
Y	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>								
Z	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>								
U	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>								

4.3.6 ステータス画面

ステータス画面では、RR0, RR1, RR2, RR4, RR5（ステータスレジスタ、インプットレジスタ）の現在の値を一定間隔でMCX304から読み出し、画面に表示します。データの読み出し間隔は、50ミリ秒間隔です。RR3 については、割り込み発生時、割り込み画面に表示します。

	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
RR0	U-HOM Z-HOM Y-HOM X-HOM U-ERR Z-ERR Y-ERR X-ERR U-DRV Z-DRV Y-DRV X-DRV															
	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
RR1	EMG ALARM LMT- LMT+ STOP2 STOP1 STOP0 ADSND ACNST AASND DSND CNST ASND CMP- CMP+															
X	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●
Y	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●
Z	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●
U	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●
RR2	HMST4 HMST3 HMST2 HMST1 HMST0 HOME EMG ALARM HLMT- HLMT+ SLMT- SLMT+															
X	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Y	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Z	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
U	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
RR4	Y-ALM Y-INP Y-EX- Y-EX+ Y-ST2 Y-ST1 Y-ST0 X-ALM X-INP X-EX- X-EX+ EMG X-ST2 X-ST1 X-ST0															
	●	●	●	●	○	●	●	●	●	●	●	●	●	●	●	●
RR5	U-ALM U-INP U-EX- U-EX+ U-ST2 U-ST1 U-ST0 Z-ALM Z-INP Z-EX- Z-EX+ Z-ST2 Z-ST1 Z-ST0															
	●	●	●	●	○	●	●	●	●	●	●	●	○	●	●	●

※ RR3は、割り込み発生時、割り込み画面に表示します。

4.3.7 Port A, B, C 出力画面

Port A, B, C 出力画面では、MC8082P, またはMC8080Pボードの汎用出力ポートA, B, Cに対して、汎用出力を行います。本画面は、MC8082P, MC8080Pボードの場合に使用できます。

	D7	D6	D5	D4	D3	D2	D1	D0
Port A	○	○	●	○	○	○	○	●
Port B	○	●	○	○	○	○	●	○
Port C	●	○	○	○	○	●	○	○

Read

4.4 MCX314As評価ツール

MCX314As 評価ツールプログラムは、MCX314Asを搭載しているボード(MC8043P, MC8043Pe)を評価するツールです。弊社ホームページよりダウンロードすることができます。(MC8000Pデバイスドライバソフトに添付) 評価ツールを実行する前に、MC8000Pデバイスドライバをインストールして下さい。

注：この章では、MCX314As 評価ツールの概要について説明します。詳細については、Tool¥MCX314As Boardフォルダの ReadMe.txt(操作説明書)を参照して下さい。

4.4.1 実行プログラムについて

実行プログラムはMCX314As-A.exeです。

■評価するMCX314Asについて

各評価ツールが評価するMCX314Asは下記の通りです。

●MCX314As-A.exe

①MCX314Asを1つ搭載しているボード (MC8043P, MC8043Pe) のMCX314As

■実行について

異なるボードに対して複数のMCX314As-A.exeを同時に実行する事ができます。exeの名前を異なる名前に変更してから実行して下さい。(例：MCX314As-A.exe、MCX314As-B.exe など)

同じボードのICに対して、複数のMCX314As-A.exeを同時に実行することはできません。

4.4.2 機能概要

評価ツールを起動すると、ボード番号選択画面が表示され、ボード番号 (ボードのロータリースイッチ番号(0~F)) を選択することができます。ボード名表示ボタンを押すと、ボード番号の横にボード名(本ドライバを使用しているボードのみ)を表示することができます。ボード番号を選択後、OKボタンを押すと、メイン画面が表示されます。

メイン画面では、各軸パラメータ設定、ドライブ命令等の命令実行、現在位置・現在速度の表示、割り込み画面表示、パラメータ・モード設定値の保存、読み出し等を行います。また、3種類のモード設定画面やステータス画面を開き、モード設定、ステータス参照等を行う事ができます。

4.4.3 メイン画面

メイン画面では、下図のような操作を行います。

ドライブ中の現在位置や
現在ドライブ速度等の表示

各軸のドライブ命令等の命令実行

位置カウンタ設定

各軸パラメータ設定

補間命令実行

モード設定画面(3個)
ステータス画面の起動

X軸ドライブ速度の軌跡表示
(プロット時のみ)

各パラメータ、モード設定値の
保存、読み出し

割り込み

割り込み発生 このRR3は、Nmc_ReadEvent 関数で読み出しています。

RR3	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	SYNC HMEND D-END C-STA C-END P≥C+ P<C+ P<C- P≥C- PULSE															
X	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○
Y	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○
Z	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○
U	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○

割り込み発生時に
割り込み画面表示

4.4.4 モード設定画面

モード設定画面では、MCX314AsのWR1～WR5（モードレジスタ、アウトプットレジスタ）を設定します。

		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
WR1		D-END	C-STA	C-END	P \geq C+	P<C+	P<C-	P \geq C-	PULSE	IN3E	IN3L	IN2E	IN2L	IN1E	IN1L	IN0E	IN0L	
X		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Y		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Z		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
U		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
WR2		INP-E	INP-L	ALM-E	ALM-L	PIND1	PIND0	PINMD	DIR-L	PLS-L	PLSMD	CMPPL	HLMT-	HLMT+	LMTMD	SLMT-	SLMT+	
X		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>						
Y		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>						
Z		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>						
U		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>						
WR3							OUT7	OUT6	OUT5	OUT4	OUTSL							
X							<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>							
Y							<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>							
Z							<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>							
U							<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>							
WR4		UOUT3	UOUT2	UOUT1	UOUT0	ZOUT3	ZOUT2	ZOUT1	ZOUT0	YOUT3	YOUT2	YOUT1	YOUT0	XOUT3	XOUT2	XOUT1	XOUT0	
X		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Y		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Z		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
U		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
WR5		BPINT	CIINT	CMPLS		EXPLS	SPD1		SPD0	AX31			AX30	AX21	AX20	AX11	AX10	
X		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input checked="" type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Y		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Z		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
U		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

4.4.5 拡張モード設定画面

拡張モード設定画面では、MCX314Asの拡張モードレジスタ (EM6, EM7) を設定します。

		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
EM6		FL2	FL1	FL0	FE4	FE3	FE2	FE1	FE0	SMODE	HMINT		VRING	AVTRI	POINV	EPINV	EPCLR
X		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>											
Y		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>											
Z		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>											
U		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>											
EM7		DCCW2	DCCW1	DCCW0	DCC-L	DCC-E	LIMIT	SAND	PCLR	ST4-D	ST4-E	ST3-D	ST3-E	ST2-D	ST2-E	ST1-D	ST1-E
X		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Y		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Z		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
U		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

4.4.6 同期動作モード設定画面

同期動作モード設定画面では、MCX314Asの同期動作モードレジスタ (SM6, SM7) を設定します。

		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
SM6		AXIS3	AXIS2	AXIS1				CMD	LPRD	IN3DW	IN3UP	D-END	D-STA	P \geq C-	P<C-	P<C+	P \geq C+
X		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
Y		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
Z		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
U		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
SM7		INT	OUT	VLSET			OPSET	EPSET	LPSET	EPSAV	LPSAV	ISTOP	SSTOP	CDRV-	CDRV+	FDRV-	FDRV+
X		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
Y		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
Z		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
U		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					

4.4.7 ステータス画面

ステータス画面では、RR0, RR1, RR2, RR4, RR5（ステータスレジスタ、インプットレジスタ）の現在の値を一定間隔でMCX314Asから読み出し、画面に表示します。データの読み出し間隔は、50ミリ秒間隔です。
RR3 については、割り込み発生時、割り込み画面に表示します。

		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
RR0		BPSC1	BPSC0	ZONE2	ZONE1	ZONE0	GNEXT	I-DRV	U-ERR	Z-ERR	Y-ERR	X-ERR	U-DRV	Z-DRV	Y-DRV	X-DRV	
		○	○	○	○	○	○	○	○	○	○	○	○	●	●	●	●
RR1		EMG	ALARM	LMT-	LMT+	IN3	IN2	IN1	IN0	ADSND	ACNST	AASND	DSND	CNST	ASND	CMP-	CMP+
X		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Y		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Z		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
U		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
RR2		HMST4 HMST3 HMST2 HMST1 HMST0 HOME							EMG ALARM HLMT- HLMT+ SLMT- SLMT+								
X		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Y		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Z		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
U		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
RR4		Y-ALM	Y-INP	Y-EX-	Y-EX+	Y-IN3	Y-IN2	Y-IN1	Y-IN0	X-ALM	X-INP	X-EX-	X-EX+	X-IN3	X-IN2	X-IN1	X-IN0
		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
RR5		U-ALM	U-INP	U-EX-	U-EX+	U-IN3	U-IN2	U-IN1	U-IN0	Z-ALM	Z-INP	Z-EX-	Z-EX+	Z-IN3	Z-IN2	Z-IN1	Z-IN0
		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

※ RR3は、割り込み発生時、割り込み画面に表示します。

5. MC8500Pシリーズ ボード

この章では、以下のボードで使用できるAPIについて説明します。

ボード	搭載 I C	I C 数	軸数
MC8541P / MC8541Pe	MCX514	1	4
MC8581P / MC8581Pe	MCX514	2	8

5.1 API

MC8000P.SYS、MC8000P.DLL がアプリケーションに提供するAPI

5.1.1 関数一覧

下表は、API関数の一覧表です。

「VC」「VB.NET」の欄は、各言語において各関数を使用できるかどうかを記載しています。

VC++の場合 [VC] の項目を参照して下さい。

VB.NETの場合 [VB.NET] の項目を参照して下さい。

C#の場合 [C#] の項目を参照して下さい。

○は使用できます。×は使用できません。

(1) 基本関数

関数名	説明	VC	VB.NET	C#	ページ	備考
Nmc_Open	ボードの使用を開始する	○	○	○	100	
Nmc_Close	ボードの使用を終了する	○	○	○	〃	
Nmc_CloseAll	全てのボードの使用を終了する	○	○	○	〃	
Nmc_GetBoardInfo	オープンしたボードの情報を取得する	○	○	○	101	
Nmc_OutPort	出力ポートにデータを書く	○	○	○	〃	
Nmc_InPort	入力ポートからデータを読む	○	○	○	〃	
Nmc_WriteReg	ボードのレジスタにデータを書き込む	○	○	○	102	
Nmc_ReadReg	ボードのレジスタからデータを読み込む	○	○	○	〃	
Nmc_SetEvent	割込みを処理するユーザー関数を設定する	○	×	○	103	
Nmc_ResetEvent	割込みを処理するユーザー関数の設定を解除する	○	×	○	〃	
Nmc_ReadEvent	割込み発生直後の各軸RR1の値を取得する	○	×	○	104	

(2) リセット、命令

関数名	説明	VC	VB.NET	C#	ページ	備考
Nmc_Reset	ボードに搭載しているICをリセットする	○	○	○	104	
Nmc_Command	指定軸の命令を実行する	○	○	○	105	
Nmc_Command_IP	補間命令を実行する	○	○	○	〃	

(3) ライトレジスタ

関数名	説明	VC	VB.NET	C#	ページ	備考
Nmc_WriteReg0	WR0 (コマンドレジスタ) 書き込み	○	○	○	105	
Nmc_WriteReg1	WR1 (モードレジスタ1) 書き込み	○	○	○	106	
Nmc_WriteReg2	WR2 (モードレジスタ2) 書き込み	○	○	○	〃	
Nmc_WriteReg3	WR3 (モードレジスタ3) 書き込み	○	○	○	〃	
Nmc_WriteReg4	WR4 (アウトプットレジスタ1) 書き込み	○	○	○	107	
Nmc_WriteReg5	WR5 (アウトプットレジスタ2) 書き込み	○	○	○	〃	
Nmc_WriteReg6	WR6 (ライトデータレジスタ1) 書き込み	○	○	○	〃	
Nmc_WriteReg7	WR7 (ライトデータレジスタ2) 書き込み	○	○	○	108	

(4) リードレジスタ

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_ReadReg0	RR0(主ステータスレジスタ) 読み出し	○	○	○	108	
Nmc_ReadReg2	RR2(ステータスレジスタ2) 読み出し	○	○	○	"	
Nmc_ReadReg3	RR3(ステータスレジスタ3) 読み出し (ページ指定なし)	○	○	○	109	
Nmc_ReadReg3P	RR3(ステータスレジスタ3) 読み出し (ページ指定あり)	○	○	○	"	
Nmc_ReadReg4	RR4(インプットレジスタ1) 読み出し	○	○	○	"	
Nmc_ReadReg5	RR5(インプットレジスタ2) 読み出し	○	○	○	110	
Nmc_ReadReg6	RR6(リードデータレジスタ1) 読み出し	○	○	○	"	
Nmc_ReadReg7	RR7(リードデータレジスタ2) 読み出し	○	○	○	"	

(5) パラメータ設定

関数名	説明	VC	VB. NET	C#		備考
Nmc_Jerk	加速度増加率設定(加加速度)	○	○	○	111	
Nmc_DJerk	減速度増加率設定	○	○	○	"	
Nmc_Acc	加速度設定	○	○	○	"	
Nmc_Dec	減速度設定	○	○	○	112	
Nmc_StartSpd	初速度設定	○	○	○	"	
Nmc_Speed	ドライブ速度設定	○	○	○	"	
Nmc_Pulse	移動パルス数/補間終点設定(VC, C#用)	○	×	○	113	
Nmc_Pulse_VB	移動パルス数/補間終点設定(VB. NET用)	×	○	×	"	
Nmc_DecP	マニュアル減速点設定(VC, C#用)	○	×	○	114	
Nmc_DecP_VB	マニュアル減速点設定(VB. NET用)	×	○	×	"	
Nmc_Center	円弧中心点設定	○	○	○	"	
Nmc_Lp	論理位置カウンタ設定	○	○	○	115	
Nmc_Ep	実位置カウンタ設定	○	○	○	"	
Nmc_CompP	ソフトリミット+レジスタ設定	○	○	○	"	
Nmc_CompM	ソフトリミット-レジスタ設定	○	○	○	116	
Nmc_AccOfst	加速カウンタオフセット設定	○	○	○	"	
Nmc_HomeSpd	原点検出速度設定	○	○	○	117	
Nmc_LpMax	論理位置カウンタ最大値設定	○	○	○	"	
Nmc_RpMax	実位置カウンタ最大値設定	○	○	○	"	
Nmc_MR0	多目的レジスタ0設定	○	○	○	118	
Nmc_MR1	多目的レジスタ1設定	○	○	○	"	
Nmc_MR2	多目的レジスタ2設定	○	○	○	"	
Nmc_MR3	多目的レジスタ3設定	○	○	○	119	
Nmc_SpeedInc	速度増減値設定	○	○	○	"	
Nmc_Timer	タイマー値設定	○	○	○	"	
Nmc_TPMax	補間・終点最大値設定	○	○	○	120	
Nmc_HLNumber	ヘリカル回転数設定	○	○	○	"	
Nmc_HLValue	ヘリカル演算値設定	○	○	○	"	

(6) その他のモード設定

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_MRmMode	多目的レジスタモード設定	○	○	○	121	
Nmc_PIO1Mode	P I O信号設定1	○	○	○	"	
Nmc_PIO2Mode	P I O信号設定2・その他設定	○	○	○	"	
Nmc_HMSrch1Mode	自動原点出しモード設定1	○	○	○	122	
Nmc_HMSrch2Mode	自動原点出しモード設定2	○	○	○	"	
Nmc_FilterMode	入力信号フィルタモード設定	○	○	○	123	
Nmc_Sync0Mode	同期動作S Y N C 0設定	○	○	○	"	
Nmc_Sync1Mode	同期動作S Y N C 1設定	○	○	○	"	
Nmc_Sync2Mode	同期動作S Y N C 2設定	○	○	○	124	
Nmc_Sync3Mode	同期動作S Y N C 3設定	○	○	○	"	
Nmc_IPMode	補間モード設定	○	○	○	"	

(7) データ読み出し

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_ReadLp	論理位置カウンタ読み出し	○	○	○	125	
Nmc_ReadEp	実位置カウンタ読み出し	○	○	○	〃	
Nmc_ReadSpeed	現在ドライブ速度読み出し	○	○	○	〃	
Nmc_ReadAccDec	現在加/減速度読み出し	○	○	○	126	
Nmc_ReadMR0	多目的レジスタ0読み出し	○	○	○	〃	
Nmc_ReadMR1	多目的レジスタ1読み出し	○	○	○	〃	
Nmc_ReadMR2	多目的レジスタ2読み出し	○	○	○	127	
Nmc_ReadMR3	多目的レジスタ3読み出し	○	○	○	〃	
Nmc_ReadCT	現在タイマー値読み出し	○	○	○	〃	
Nmc_ReadTX	補間・終点最大値読み出し	○	○	○	128	
Nmc_ReadCHLN	現在ヘリカル回転数読み出し	○	○	○	〃	
Nmc_ReadHLV	ヘリカル演算値読み出し	○	○	○	〃	
Nmc_ReadWR1	WR1設定値読み出し	○	○	○	129	
Nmc_ReadWR2	WR2設定値読み出し	○	○	○	〃	
Nmc_ReadWR3	WR3設定値読み出し	○	○	○	〃	
Nmc_ReadMRM	多目的レジスタモード設定読み出し	○	○	○	130	
Nmc_ReadP1M	PIO信号設定1読み出し	○	○	○	〃	
Nmc_ReadP2M	PIO信号設定2その他設定読み出し	○	○	○	〃	
Nmc_ReadAc	加速度設定値読み出し	○	○	○	131	
Nmc_ReadStartSpd	初速度設定値読み出し	○	○	○	〃	
Nmc_ReadSetSpeed	ドライブ速度設定値読み出し	○	○	○	〃	
Nmc_ReadPulse	移動パルス数/終点設定値読み出し	○	○	○	132	

(8) 状態取得

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_GetDriveStatus	ドライブ状態取得	○	○	○	132	
Nmc_GetCNextStatus	連続補間次データ書込み可能状態取得	○	○	○	133	
Nmc_GetSc	連続補間プリバッファスタックカウンタ取得	○	○	○	〃	

(9) 書き込み・読み出し

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_WriteRegSetAxis	軸指定ライトレジスタ書き込み(WR1~3)	○	○	○	134	
Nmc_ReadRegSetAxis	軸指定リードレジスタ読み出し(RR2~3)	○	○	○	〃	
Nmc_WriteData	データ書き込み(パラメータ)	○	○	○	135	
Nmc_ReadData	データ読み出し(4バイト読み出し)	○	○	○	〃	

(10) B P補間・連続補間実行

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_2BPExecMC8500P	2軸B P補間実行	○	○	○	136	
Nmc_3BPExecMC8500P	3軸B P補間実行	○	○	○	138	
Nmc_4BPExecMC8500P	4軸B P補間実行	○	○	○	140	
Nmc_2BPExecMC8500P_BG	2軸B P補間実行(バックグラウンドで実行)	○	○	○	142	
Nmc_3BPExecMC8500P_BG	3軸B P補間実行(バックグラウンドで実行)	○	○	○	145	
Nmc_4BPExecMC8500P_BG	4軸B P補間実行(バックグラウンドで実行)	○	○	○	148	
Nmc_2CIPExecMC8500P	2軸連続補間実行	○	○	○	151	
Nmc_3CIPExecMC8500P	3軸連続補間実行	○	○	○	153	
Nmc_4CIPExecMC8500P	4軸連続補間実行	○	○	○	155	
Nmc_2CIPExecMC8500P_BG	2軸連続補間実行(バックグラウンドで実行)	○	○	○	157	
Nmc_3CIPExecMC8500P_BG	3軸連続補間実行(バックグラウンドで実行)	○	○	○	160	
Nmc_4CIPExecMC8500P_BG	4軸連続補間実行(バックグラウンドで実行)	○	○	○	163	
Nmc_IPStop	補間実行を中断する	○	○	○	166	
Nmc_IPGetMsgNo	補間終了時の受信メッセージからボード番号とIC番号取得	○	○	○	〃	

(11) ヘリカル補間実行

関数名	説明	VC	VB. NET	C#	ページ	備考
Nmc_HLValueExe	ヘリカル演算実行	○	○	○	167	
Nmc_HLExec	ヘリカル補間実行	○	○	○	169	

5.1.2 関数仕様

VC++の場合 : VC と[VC]の項目を参照して下さい。
 VB.NETの場合 : VB.NET と[VB.NET]の項目を参照して下さい。
 C#の場合 : C#と[C#]の項目を参照して下さい。

指定のない項目は、各言語共通の内容です。

関数名	機能 及び 内容
Nmc_Open	<p>ボードの使用を開始する。</p> <p>VC BOOL Nmc_Open(int No, BOOL IntrptFlg); VB.NET Function Nmc_Open(ByVal No As Integer, ByVal IntrptFlg As Integer) As Integer C# bool MC8000P.Nmc_Open(int No, bool IntrptFlg);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IntrptFlg 割り込みを使用するかどうかを指定する。 [VC] TRUE: 使用する。FALSE: 使用しない。 [VB.NET] False固定。割り込みを使用しない設定。(VBでは割り込みを使用できません) [C#] true : 使用する。false : 使用しない。</p> <p>戻り値 [VC] オープンに成功するとTRUE、失敗するとFALSE [VB.NET] オープンに成功すると0以外、失敗すると0 [C#] オープンに成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] status = Nmc_Open(0, FALSE); // ボード番号0をオープン、割り込みを使用しない [VB.NET] status = Nmc_Open(0, False) [C#] status = MC8000P.Nmc_Open(0, false);</p>
Nmc_Close	<p>ボードの使用を終了する。</p> <p>VC BOOL Nmc_Close(int No); VB.NET Function Nmc_Close(ByVal No As Integer) As Integer C# bool MC8000P.Nmc_Close(int No);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>戻り値 [VC] クローズに成功するとTRUE、失敗するとFALSE [VB.NET] クローズに成功すると0以外、失敗すると0 [C#] クローズに成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] status = Nmc_Close(0); // ボード番号0をクローズ [VB.NET] status = Nmc_Close(0) [C#] status = MC8000P.Nmc_Close(0);</p>
Nmc_CloseAll	<p>全てのボードの使用を終了する。</p> <p>VC BOOL Nmc_CloseAll(void); VB.NET Function Nmc_CloseAll() As Integer C# bool MC8000P.Nmc_CloseAll();</p> <p>入力パラメータ なし</p> <p>戻り値 [VC] クローズに成功するとTRUE、失敗するとFALSE [VB.NET] クローズに成功すると0以外、失敗すると0 [C#] クローズに成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] status = Nmc_CloseAll(); // 全てのボードをクローズ [VB.NET] status = Nmc_CloseAll() [C#] status = MC8000P.Nmc_CloseAll();</p>

関数名	機能 及び 内容
Nmc_GetBoardInfo	<p>オープンしたボードの情報としてデバイスIDを取得する。</p> <pre> VC BOOL Nmc_GetBoardInfo(int No, USHORT* pDeviceID); VB.NET Function Nmc_GetBoardInfo(ByVal No As Integer, ByRef DeviceID As Short) As Integer C# bool MC8000P.Nmc_GetBoardInfo(int No, out ushort DeviceID); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) DeviceID [VC] 取得したボードのデバイスIDを格納する変数のアドレス [VB.NET][C#] 取得したボードのデバイスIDを格納する変数 各ボードのデバイスIDは「4.1.3 補足説明」(1)③参照。 [VC][VB] MC8541P/MC8541PeはID_MC8541P、MC8581P/MC8581PeはID_MC8581P。 [C#] MC8541P/MC8541PeはDev_ID.MC8541P、MC8581P/MC8581PeはDev_ID.MC8581P を指定する。</p> <p>戻り値 [VC] 取得が成功するとTRUE、失敗するとFALSE [VB.NET] 取得が成功すると0以外、失敗すると0 [C#] 取得が成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] USHORT DeviceID; status = Nmc_GetBoardInfo(No, &DeviceID); // デバイスID取得 if(DeviceID == ID_MC8581P) // MC8581Pの場合 [VB.NET] Dim DeviceID As Short status = Nmc_GetBoardInfo(No, DeviceID) If DeviceID = ID_MC8581P Then [C#] ushort DeviceID; status = Nmc_GetBoardInfo(No, out DeviceID); if(DeviceID == Dev_ID.MC8581P) </p>
	<p>出力ポートに2バイトデータを書き込む。</p> <pre> VC void Nmc_OutPort(int No, long Adr, long Dat); VB.NET Sub Nmc_OutPort(ByVal No As Integer, ByVal adr As Integer, ByVal Dat As Integer) C# void MC8000P.Nmc_OutPort(int No, REG_MCX Adr, int Dat); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) Adr 書き込むアドレス。各ボード取扱説明書に記載しているI/Oアドレス。 詳細は「5.1.3 補足説明」(1)①、(6)参照。 Dat 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_OutPort(No, 0, MC8500P_CMD_RST); // I C - Aのコマンドリセット (WRO書き込み) [VB.NET] Call Nmc_OutPort(No, 0, MC8500P_CMD_RST) [C#] MC8000P.Nmc_OutPort(No, REG_MCX.WRO_A, MC8500P_CMD_RST); </p>
Nmc_InPort	<p>入力ポートから2バイトデータを読み出す。</p> <pre> VC long Nmc_InPort(int No, long Adr); VB.NET Function Nmc_InPort(ByVal No As Integer, ByVal adr As Integer) As Integer C# int MC8000P.Nmc_InPort(int No, REG_MCX Adr); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) Adr 読み出すアドレス。各ボード取扱説明書に記載しているI/Oアドレス。 詳細は「5.1.3 補足説明」(1)①、(6)参照。</p> <p>戻り値 入力ポートから読み込んだデータ</p> <p>使用例 [VC] data = Nmc_InPort(No, 0); // I C - Aのリードレジスタ RRO の読み出し [VB.NET] data = Nmc_InPort(No, 0) [C#] data = MC8000P.Nmc_InPort(No, REG_MCX.RRO_A); <p>注意 [VC][C#] RR1レジスタ(ステータスレジスタ:割り込みが発生した要因を表示するレジスタ)のデータ 読み出しに関しては、Nmc_ReadEvent関数の説明を参考にして下さい。</p> </p>

関数名	機能 及び 内容
Nmc_WriteReg	<p>ボードのライトレジスタ (WR0~WR7) にデータを書き込む。</p> <pre> VC void Nmc_WriteReg(int No, int IcNo, long RegNum, long Dat); VB.NET Sub Nmc_WriteReg(ByVal No As Integer, ByVal IcNo As Integer, ByVal RegNum As Integer, ByVal Dat As Integer) C# void MC8000P.Nmc_WriteReg(int No, int IcNo, int RegNum, int Dat); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15))</p> <p>IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>RegNum 書き込むレジスタ (MCX_WR0~MCX_WR7)。詳細は「5.1.3 補足説明」(1)参照。 例) [VC] [VB.NET] WR0の場合は MCX_WR0 を、WR1の場合は MCX_WR1 を指定する。 [C#] WR0の場合は REG_MCX.WR0 を、WR1の場合は REG_MCX.WR1 を指定する。</p> <p>Dat 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例</p> <pre> [VC] Nmc_WriteReg(No, 0, MCX_WR0, MC8500P_CMD_RST); // IC-Aのコマンドリセット (WR0書き込み) [VB.NET] Call Nmc_WriteReg(No, 0, MCX_WR0, MC8500P_CMD_RST) [C#] MC8000P.Nmc_WriteReg(No, 0, REG_MCX.WR0, MC8500P_CMD_RST); </pre>
Nmc_ReadReg	<p>ボードのリードレジスタ (RR0~RR7) からデータを読み出す。</p> <pre> VC long Nmc_ReadReg(int No, int IcNo, long RegNum); VB.NET Function Nmc_ReadReg(ByVal No As Integer, ByVal IcNo As Integer, ByVal RegNum As Integer) As Integer C# void MC8000P.Nmc_ReadReg(int No, int IcNo, int RegNum); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15))</p> <p>IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>RegNum 読み出すレジスタ (MCX_RR0~MCX_RR7)。詳細は「5.1.3 補足説明」(1)参照。 例) [VC] [VB.NET] RR0の場合は MCX_RR0 を、RR2の場合は MCX_RR2 を指定する。 [C#] RR0の場合は REG_MCX.RR0 を、RR1の場合は REG_MCX.RR1 を指定する。</p> <p>戻り値 リードレジスタから読み出したデータ</p> <p>使用例</p> <pre> [VC] data = Nmc_ReadReg(No, 0, MCX_RR0); // IC-Aのリードレジスタ RR0 の読み出し [VB.NET] data = Nmc_ReadReg(No, 0, MCX_RR0) [C#] data = MC8000P.Nmc_ReadReg(No, 0, REG_MCX.WR0); </pre> <p>注意</p> <pre> [VC] [C#] RR1レジスタのデータ読み出しに関しては、Nmc_ReadEvent関数の説明を参考にして下さい。 </pre>

関数名	機能 及び 内容
Nmc_SetEvent	<p>割り込みを処理するユーザー関数を設定する。 この関数を実行すると、割り込みが発生した時にユーザー関数が呼び出され、指定した引数が1つ渡されます。 このユーザー関数は1つのスレッドとして起動されます。 割り込み処理する関数の設定を解除する場合は Nmc_ResetEvent を実行して下さい。</p> <p>VC BOOL Nmc_SetEvent(int No, LPTHREAD_START_ROUTINE UserFunc, LPVOID lpParameter); VB.NET 使用できません C# bool MC8000P.Nmc_SetEvent(int No, UserThread Callback, int param);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15)) [VC]UserFunc ユーザー関数のアドレス [VC]lpParameter ユーザー関数スレッドに渡す1つの引数を指定する。 引数を使用しない場合は、NULL等で良い。 スレッドで使用可能なポインタを設定して下さい。 ポインタの場合、ユーザー関数呼び出し時に使用可能なポインタを設定して下さい。 [C#] Callback ユーザーメソッド (デリゲート型) 詳細は「4.1.4使用方法」を参照。 [C#] param 割り込み発生時にユーザー関数に渡す1つのパラメータ(数値等int限定)を指定します。</p> <p>戻り値 成功するとTRUE、失敗するとFALSE</p> <p>使用例</p> <p>[VC] (ボード番号0の場合) status = Nmc_SetEvent(0, MC_EventFunc0, lpParam); // 関数のアドレスと引数を設定 Nmc_WriteReg1(0, 0, AXIS_ALL, 0x0080); // IC-Aのドライブ終了割り込み発生 (全軸)</p> <p>(ボード番号1の場合) status = Nmc_SetEvent(1, MC_EventFunc1, NULL); //関数のアドレスと引数を設定 Nmc_WriteReg1(1, 0, AXIS_ALL, 0x0080); // IC-Aのドライブ終了割り込み発生 (全軸)</p> <p>■割り込みユーザー関数例</p> <pre> DWORD WINAPI MC_EventFunc0(LPVOID lpParam) { long Rr1X, Rr1Y, Rr1Z, Rr1U; Nmc_ReadEvent(0, 0, &Rr1X, &Rr1Y, &Rr1Z, &Rr1U); // ボード0, IC-AのRR1割り込みデータ読み出し return 0; } DWORD WINAPI MC_EventFunc1(LPVOID lpParam) { long Rr1X, Rr1Y, Rr1Z, Rr1U; Nmc_ReadEvent(1, 0, &Rr1X, &Rr1Y, &Rr1Z, &Rr1U); // ボード1, IC-AのRR1割り込みデータ読み出し return 0; } </pre> <p>[C#] // 割り込みユーザーメソッド isr をデリゲート型変数に代入 MC8000P.callback[0] = new MC8000P.UserThread(isr); // 割り込みユーザーメソッドの設定 MC8000P.Nmc_SetEvent(no, MC8000P.callback[0], param);</p>
Nmc_ResetEvent	<p>割り込みを処理するユーザー関数の設定を解除する。 この関数を実行すると、割り込みが発生してもユーザー関数は呼び出されません。</p> <p>VC BOOL Nmc_ResetEvent(int No); VB.NET 使用できません C# bool MC8000P.Nmc_ResetEvent(int No);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15))</p> <p>戻り値 [VC] 成功するとTRUE、失敗するとFALSE [C#] 成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] status = Nmc_ResetEvent(No); [C#] status = MC8000P.Nmc_ResetEvent(No);</p>

関数名	機能 及び 内容
Nmc_ReadEvent	<p>割込み発生直後の各軸RR1の値を取得する。(ドライバ内のRR1データは読み出し後クリアされる)</p> <p>VC BOOL Nmc_ReadEvent(int No, int IcNo, long* RrIrX, long* RrIrY, long* RrIrZ, long* RrIrU); VB.NET 使用できません C# bool MC8000P.Nmc_ReadEvent(int No, int IcNo, out int RrIrX, out int RrIrY, out int RrIrZ, out int RrIrU);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 RrIrX X軸のRR1を格納する為のバッファへのポインタ RrIrY Y軸のRR1を格納する為のバッファへのポインタ RrIrZ Z軸のRR1を格納する為のバッファへのポインタ RrIrU U軸のRR1を格納する為のバッファへのポインタ</p> <p>戻り値 [VC] 成功するとTRUE、失敗するとFALSE [C#] 成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] long Rr1X[2], Rr1Y[2], Rr1Z[2], Rr1U[2]; Nmc_ReadEvent(No, 0, &Rr1X[0], &Rr1Y[0], &Rr1Z[0], &Rr1U[0]); // IC-AのRR1データを読み出す Nmc_ReadEvent(No, 1, &Rr1X[1], &Rr1Y[1], &Rr1Z[1], &Rr1U[1]); // IC-BのRR1データを読み出す [C#] int Rr1X, Rr1Y, Rr1Z, Rr1U; // X軸、Y軸、Z軸、U軸 MC8000P.Nmc_ReadEvent(No, IcNo, out Rr1X, out Rr1Y, out Rr1Z, out Rr1U);</p> <p>注意 ボードで割込みが発生した直後ドライバ内でRR1を読み出してしまいうのでRR1はクリアされてしまいます。割込み発生直後のRR1を確認する場合はこの関数を使用してください。 また、Nmc_SetEvent、Nmc_ResetEvent関数の実行とは関係なく、割込みが発生するとドライバは必ずRR1データを読み出し保存します。ドライバ内に保存されたRR1データは、Nmc_ReadEvent関数を実行して読み出すとクリアされます。 ドライバ内のRR1データをクリアしたい時は、Nmc_ReadEvent関数を実行して下さい。</p>
Nmc_Reset	<p>ボードに搭載しているICをリセットする。</p> <p>VC void Nmc_Reset(int No, int IcNo); VB.NET Sub Nmc_Reset(ByVal No As Integer, ByVal IcNo As Integer) C# void MC8000P.Nmc_Reset(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Reset(1, 0); // ボード番号1のIC-Aをリセットする [VB.NET] Call Nmc_Reset(1, 0) [C#] MC8000P.Nmc_Reset(1, 0);</p>

関数名	機能 及び 内容
Nmc_Command	<p>指定軸の命令を実行する。(WROに指定軸の命令を書く)</p> <pre> VC void Nmc_Command(int No, int IcNo, int axis, int cmd); VB.NET Sub Nmc_Command(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal cmd As Integer) C# void MC8000P.Nmc_Command(int No, int IcNo, AXIS axis, CMD cmd); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis 命令を実行する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 cmd 命令番号。定義ファイル(※1)内のコマンド定義の「ドライブ命令、その他の命令」の中から1つを 指定する。相対位置ドライブの場合はMC8500P_CMD_DRVRLを指定する。 [C#]の場合は「5.1.3 補足説明」(1)④参照 ※1 : [VC]MC8000P_DLL.H / [VB.REMAINING]MC8000P_DLL.vb</p> <p>戻り値 なし</p> <p>使用例</p> <pre> [VC] Nmc_Command(No, IcNo, AXIS_X, MC8500P_CMD_DRVRL); // X軸の相対位置ドライブを実行する [VB.NET] Call Nmc_Command(No, IcNo, AXIS_X, MC8500P_CMD_DRVRL) [C#] MC8000P.Nmc_Command(No, IcNo, AXIS.X, CMD.MC8500P_CMD_DRVRL); </pre>
Nmc_Command_IP	<p>補間命令を実行する。(WROに補間命令を書く)</p> <pre> VC void Nmc_Command_IP(int No, int IcNo, int cmd); VB.NET Sub Nmc_Command_IP(ByVal No As Integer, ByVal IcNo As Integer, ByVal cmd As Integer) C# void MC8000P.Nmc_Command_IP(int No, int IcNo, CMD cmd); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 cmd 命令番号。定義ファイル(※1)内のコマンド定義の「補間命令」の中から1つを指定する。 2軸直線補間ドライブの場合はMC8500P_CMD_2CIPを指定する。 [C#]の場合は「5.1.3 補足説明」(1)④参照 ※1 : [VC]MC8000P_DLL.H / [VB.NET]MC8000P_DLL.vb</p> <p>戻り値 なし</p> <p>使用例</p> <pre> [VC] Nmc_IPMode(No, IcNo, 0x0003); // 補間軸設定 (主軸 : X、第2軸 : Y) Nmc_Command_IP(No, IcNo, MC8500P_CMD_2CIP); // 2軸直線補間ドライブを実行する [VB.NET] Call Nmc_IPMode(No, IcNo, &H3) Call Nmc_Command_IP(No, IcNo, MC8500P_CMD_2CIP) [C#] MC8000P.Nmc_IPMode(No, IcNo, 0x0003); MC8000P.Nmc_Command_IP(No, IcNo, CMD.MC8500P_CMD_2CIP); </pre>
Nmc_WriteReg0	<p>WRO (コマンドレジスタ)にデータを書き込む。</p> <pre> VC void Nmc_WriteReg0(int No, int IcNo, long wdata); VB.NET Sub Nmc_WriteReg0(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg0(int No, int IcNo, int wdata); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例</p> <pre> [VC] Nmc_WriteReg0(No, IcNo, 0x0150); // X軸の相対位置ドライブを実行する [VB.NET] Call Nmc_WriteReg0(No, IcNo, &H150) [C#] MC8000P.Nmc_WriteReg0(No, IcNo, 0x0150); </pre>

関数名	機能 及び 内容
Nmc_WriteReg1	<p>WR 1 (モードレジスタ 1) にデータを書き込む。</p> <p>VC void Nmc_WriteReg1(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_WriteReg1(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg1(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「5.1.3 補足説明」(2)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg1(No, IcNo, AXIS_X, 0x0080); // ドライブエンド割り込み発生 (X 軸) [VB.NET] Call Nmc_WriteReg1(No, IcNo, AXIS_X, &H80) [C#] MC8000P.Nmc_WriteReg1(No, IcNo, AXIS.X, 0x0080);</p>
Nmc_WriteReg2	<p>WR 2 (モードレジスタ 2) にデータを書き込む。</p> <p>VC void Nmc_WriteReg2(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_WriteReg2(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg2(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「5.1.3 補足説明」(2)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg2(No, IcNo, AXIS_Y, 0x0200); // ALARM 有効 (Y 軸) [VB.NET] Call Nmc_WriteReg2(No, IcNo, AXIS_Y, &H200) [C#] MC8000P.Nmc_WriteReg2(No, IcNo, AXIS.Y, 0x0200);</p>
Nmc_WriteReg3	<p>WR 3 (モードレジスタ 3) にデータを書き込む。</p> <p>VC void Nmc_WriteReg3(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_WriteReg3(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg3(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「5.1.3 補足説明」(2)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg3(No, IcNo, AXIS_ALL, 0x0004); // 全軸 S 字加減速設定 [VB.NET] Call Nmc_WriteReg3(No, IcNo, AXIS_ALL, &H4) [C#] MC8000P.Nmc_WriteReg3(No, IcNo, AXIS.ALL, 0x0004);</p>

関数名	機能 及び 内容
Nmc_WriteReg4	<p>WR 4 (アウトプットレジスタ 1) にデータを書き込む。</p> <p>VC void Nmc_WriteReg4(int No, int IcNo, long wdata); VB.NET Sub Nmc_WriteReg4(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg4(int No, int IcNo, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg4(No, IcNo, 0x0001); // X軸汎用出力PI00 Hiレベル出力 [VB.NET] Call Nmc_WriteReg4(No, IcNo, &H1) [C#] MC8000P.Nmc_WriteReg4(No, IcNo, 0x0001);</p>
Nmc_WriteReg5	<p>WR 5 (アウトプットレジスタ 2) にデータを書き込む。</p> <p>VC void Nmc_WriteReg5(int No, int IcNo, long wdata); VB.NET Sub Nmc_WriteReg5(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg5(int No, int IcNo, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg5(No, IcNo, 0x0001); // Z軸汎用出力PI00 Hiレベル出力 [VB.NET] Call Nmc_WriteReg5(No, IcNo, &H1) [C#] MC8000P.Nmc_WriteReg5(No, IcNo, 0x0024);</p>
Nmc_WriteReg6	<p>WR 6 (ライトデータレジスタ 1) にデータを書き込む。</p> <p>VC void Nmc_WriteReg6(int No, int IcNo, long wdata); VB.NET Sub Nmc_WriteReg6(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg6(int No, int IcNo, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg6(No, IcNo, 0x1234); // ライトデータレジスタ 1 にデータ (1234)H を書く [VB.NET] Call Nmc_WriteReg6(No, IcNo, &H1234) [C#] MC8000P.Nmc_WriteReg6(No, IcNo, 0x1234);</p>

関数名	機能 及び 内容
Nmc_WriteReg7	<p>WR 7 (ライトデータレジスタ 2) にデータを書き込む。</p> <p>VC void Nmc_WriteReg7(int No, int IcNo, long wdata); VB.NET Sub Nmc_WriteReg7(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteReg7(int No, int IcNo, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_WriteReg7(No, IcNo, 0x5678); // ライトデータレジスタ 2 にデータ (5678)H を書く [VB.NET] Call Nmc_WriteReg7(No, IcNo, &H5678) [C#] MC8000P.Nmc_WriteReg7(No, IcNo, 0x5678);</p>
Nmc_ReadReg0	<p>RR 0 (主ステータスレジスタ) のデータを読み出す。</p> <p>VC long Nmc_ReadReg0(int No, int IcNo); VB.NET Function Nmc_ReadReg0(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadReg0(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 RR 0 (主ステータスレジスタ) のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg0(No, IcNo); // RR 0 のデータを読む [VB.NET] Data = Nmc_ReadReg0(No, IcNo) [C#] Data = MC8000P.Nmc_ReadReg0(No, IcNo);</p>
Nmc_ReadReg2	<p>RR 2 (ステータスレジスタ 2) のデータを読み出す。</p> <p>VC long Nmc_ReadReg2(int No, int IcNo, int axis); VB.NET Function Nmc_ReadReg2(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadReg2(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 RR 2 (ステータスレジスタ 2) のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg2(No, IcNo, AXIS_Y); // Y軸のRR 2のデータを読む [VB.NET] Data = Nmc_ReadReg2(No, IcNo, AXIS_Y) [C#] Data = MC8000P.Nmc_ReadReg2(No, IcNo, AXIS.Y);</p>

関数名	機能 及び 内容
Nmc_ReadReg3	<p>R R 3 (ステータスレジスタ 3) のデータを読み出す。</p> <pre> VC long Nmc_ReadReg3(int No, int IcNo, int axis); VB.NET Function Nmc_ReadReg3(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000.Nmc_ReadReg3(int No, int IcNo, AXIS axis); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 R R 3 (ステータスレジスタ 3) のデータ</p> <p>使用例 [VC] Nmc_Command(No, IcNo, AXIS_Y, MC8500P_CMD_RR3P0); // Y軸のR R 3のページ0表示 Data = Nmc_ReadReg3(No, IcNo, AXIS_Y); // Y軸のR R 3のデータを読む [VB.NET] Call Nmc_Command(No, IcNo, AXIS_Y, MC8500P_CMD_RR3P0) Data = Nmc_ReadReg3(No, IcNo, AXIS_Y) [C#] MC8000P.Nmc_Command(No, IcNo, AXIS.Y, CMD.MC8500P_CMD_RR3P0); Data = MC8000P.Nmc_ReadReg3(No, IcNo, AXIS.Y);</p> <p>注意 R R 3 (ステータスレジスタ 3) は、ページ0およびページ1の2種類存在します。読み出す前にR R 3 ページ表示命令(7Ah, 7Bh)を書き込むことでページの指定をします。リセット時はページ0となります。</p>
Nmc_ReadReg3P	<p>R R 3 (ステータスレジスタ 3) のデータを読み出す。(ページ指定あり)</p> <pre> VC long Nmc_ReadReg3P(int No, int IcNo, int axis, int Page); VB.NET Function Nmc_ReadReg3P(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal Page As Integer) As Integer C# int MC8000.Nmc_ReadReg3P(int No, int IcNo, AXIS axis, int Page); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。 Page データを読み出すページ、ページ0は0, ページ1は1。</p> <p>戻り値 指定したページのR R 3 (ステータスレジスタ 3) データ</p> <p>使用例 [VC] Data = Nmc_ReadReg3P(No, IcNo, AXIS_Y, 0); // Y軸のR R 3のページ0のデータを読む [VB.NET] Data = Nmc_ReadReg3P(No, IcNo, AXIS_Y, 0) [C#] Data = MC8000P.Nmc_ReadReg3P(No, IcNo, AXIS.Y, 0);</p>
Nmc_ReadReg4	<p>R R 4 (P I Oリードレジスタ 1) のデータを読み出す。</p> <pre> VC long Nmc_ReadReg4(int No, int IcNo); VB.NET Function Nmc_ReadReg4(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadReg4(int No, int IcNo); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 R R 4 (P I Oリードレジスタ 1) のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg4(No, IcNo); // R R 4のデータを読む [VB.NET] Data = Nmc_ReadReg4(No, IcNo) [C#] Data = MC8000P.Nmc_ReadReg4(No, IcNo);</p>

関数名	機能 及び 内容
Nmc_ReadReg5	<p>RR5 (P I Oリードレジスタ2)のデータを読み出す。</p> <p>VC long Nmc_ReadReg5(int No, int IcNo); VB.NET Function Nmc_ReadReg5(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadReg5(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 RR5 (P I Oリードレジスタ2)のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg5 (No, IcNo); // RR5のデータを読む [VB.NET] Data = Nmc_ReadReg5 (No, IcNo) [C#] Data = MC8000P.Nmc_ReadReg5 (No, IcNo);</p>
Nmc_ReadReg6	<p>RR6 (リードデータレジスタ1)のデータを読み出す。</p> <p>VC long Nmc_ReadReg6(int No, int IcNo); VB.NET Function Nmc_ReadReg6(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadReg6(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 RR6 (リードデータレジスタ1)のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg6 (No, IcNo); // RR6のデータを読む [VB.NET] Data = Nmc_ReadReg6 (No, IcNo) ' RR6のデータを読む [C#] Data = MC8000P.Nmc_ReadReg6 (No, int IcNo);</p>
Nmc_ReadReg7	<p>RR7 (リードデータレジスタ2)のデータを読み出す。</p> <p>VC long Nmc_ReadReg7(int No, int IcNo); VB.NET Function Nmc_ReadReg7(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadReg7(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 RR7 (リードデータレジスタ2)のデータ</p> <p>使用例 [VC] Data = Nmc_ReadReg7 (No, IcNo); // RR7のデータを読む [VB.NET] Data = Nmc_ReadReg7 (No, IcNo) ' RR7のデータを読む [C#] Data = MC8000P.Nmc_ReadReg7 (No, IcNo);</p>

関数名	機能 及び 内容
Nmc_Jerk	<p>加速度増加率を設定する。</p> <pre> VC void Nmc_Jerk(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Jerk(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Jerk(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 Axis データを設定する軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Jerk(No, IcNo, AXIS_X, 1000); // 加速度増加率に 1000 を設定する(X軸) [VB.NET] Call Nmc_Jerk(No, IcNo, AXIS_X, 1000) [C#] MC8000P.Nmc_Jerk(No, IcNo, AXIS.X, 1000);</p>
Nmc_DJerk	<p>減速度増加率を設定する。</p> <pre> VC void Nmc_DJerk(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_DJerk(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_DJerk(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 Axis データを設定する軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_DJerk(No, IcNo, AXIS_X, 1000); // 減速度増加率に 1000 を設定する(X軸) [VB.NET] Call Nmc_DJerk(No, IcNo, AXIS_X, 1000) [C#] MC8000P.Nmc_DJerk(No, IcNo, AXIS.X, 1000);</p>
Nmc_Acc	<p>加速度を設定する。</p> <pre> VC void Nmc_Acc(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Acc(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Acc(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 Axis データを設定する軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。 詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Acc(No, IcNo, AXIS_Y, 100); // 加速度に 100 を設定する(Y軸) [VB.NET] Call Nmc_Acc(No, IcNo, AXIS_Y, 100) [C#] MC8000P.Nmc_Acc(No, IcNo, AXIS.Y, 100);</p>

関数名	機能 及び 内容
Nmc_Dec	<p>減速度を設定する。</p> <pre> VC void Nmc_Dec(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Dec(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Dec(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Dec(No, IcNo, AXIS_Z, 100); // 減速度に 100 を設定する(Z軸) [VB.NET] Call Nmc_Dec(No, IcNo, AXIS_Z, 100) [C#] MC8000P.Nmc_Dec(No, IcNo, AXIS.Z, 100);</p>
Nmc_StartSpd	<p>初速度を設定する。</p> <pre> VC void Nmc_StartSpd(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_StartSpd(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_StartSpd(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_StartSpd(No, IcNo, AXIS_U, 100); // 初速度に 100 を設定する(U軸) [VB.NET] Call Nmc_StartSpd(No, IcNo, AXIS_U, 100) [C#] MC8000P.Nmc_StartSpd(No, IcNo, AXIS.U, 100);</p>
Nmc_Speed	<p>ドライブ速度を設定する。</p> <pre> VC void Nmc_Speed(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Speed(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Speed(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 // ドライブ速度に 1000 を設定する(X, Y軸) [VC] Nmc_Speed(No, IcNo, AXIS_X AXIS_Y, 1000); [VB.NET] Call Nmc_Speed(No, IcNo, AXIS_X or AXIS_Y, 1000) [C#] MC8000P.Nmc_Speed(No, IcNo, AXIS.X AXIS.Y, 1000);</p>

関数名	機能 及び 内容
Nmc_DecP	<p>マニュアル減速点を設定する。(VC, C#専用)</p> <p>VC void Nmc_DecP(int No, int IcNo, int axis, ULONG wdata); VB.NET 使用できません C# void MC8000P.Nmc_DecP(int No, int IcNo, AXIS axis, uint wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_DecP(No, IcNo, AXIS_U, 30000); // マニュアル減速点に 30000 を設定する(U軸) [C#] MC8000P.Nmc_DecP(No, IcNo, AXIS.U, 30000);</p>
Nmc_DecP_VB	<p>マニュアル減速点を設定する。(VB.NET専用)</p> <p>VC 使用できません VB.NET Sub Nmc_DecP_VB(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Double) C# 使用できません</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VB.NET] Call Nmc_DecP_VB(No, IcNo, AXIS_X, 40000) ' マニュアル減速点に 40000 を設定する(X軸)</p>
Nmc_Center	<p>円弧中心点を設定する。</p> <p>VC void Nmc_Center(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Center(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Center(int No, int IcNo, AXIS axis, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Center(No, IcNo, AXIS_Y, 1500); // 円弧中心点に 1500 を設定する(Y軸) [VB.NET] Call Nmc_Center(No, IcNo, AXIS_Y, 1500) [C#] MC8000P.Nmc_Center(No, IcNo, AXIS.Y, 1500);</p>

関数名	機能 及び 内容
Nmc_Lp	<p>論理位置カウンタを設定する。</p> <pre> VC void Nmc_Lp(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Lp(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Lp(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Lp(No, IcNo, AXIS_ALL, 0); // 全軸の論理位置カウンタを0クリアする [VB.NET] Call Nmc_Lp(No, IcNo, AXIS_ALL, 0) [C#] MC8000P.Nmc_Lp(No, IcNo, AXIS.ALL, 0);</p>
Nmc_Ep	<p>実位置カウンタを設定する。</p> <pre> VC void Nmc_Ep(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Ep(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Ep(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Ep(No, IcNo, AXIS_ALL, 0); // 全軸の実位置カウンタを0クリアする [VB.NET] Call Nmc_Ep(No, IcNo, AXIS_ALL, 0) [C#] MC8000P.Nmc_Ep(No, IcNo, AXIS.ALL, 0);</p>
Nmc_CompP	<p>ソフトリミット+レジスタを設定する。</p> <pre> VC void Nmc_CompP(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_CompP(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_CompP(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_CompP(No, IcNo, AXIS_X, 50000); // ソフトリミット+レジスタに 50000 を設定する(× 軸) [VB.NET] Call Nmc_CompP(No, IcNo, AXIS_X, 50000) [C#] MC8000P.Nmc_CompP(No, IcNo, AXIS.X, 50000);</p>

関数名	機能 及び 内容
Nmc_CompM	<p>ソフトリミッターレジスタを設定する。</p> <pre> VC void Nmc_CompM(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_CompM(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_CompM(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_CompM(No, IcNo, AXIS_X, -50000); // ソフトリミッターレジスタに -50000 を設定する(X軸) [VB.NET] Call Nmc_CompM(No, IcNo, AXIS_X, -50000) [C#] MC8000P.Nmc_CompM(No, IcNo, AXIS.X, -50000);</p>
Nmc_AccOfst	<p>加速カウンタオフセットを設定する。</p> <pre> VC void Nmc_AccOfst(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_AccOfst(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_AccOfst(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_AccOfst(No, IcNo, AXIS_Y, 20); // 加速カウンタオフセットに 20 を設定する(Y軸) [VB.NET] Call Nmc_AccOfst(No, IcNo, AXIS_Y, 20) [C#] MC8000P.Nmc_AccOfst(No, IcNo, AXIS.Y, 20);</p>

関数名	機能 及び 内容
Nmc_HomeSpd	<p>原点検出速度を設定する。</p> <pre> VC void Nmc_HomeSpd(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_HomeSpd(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_HomeSpd(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_HomeSpd(No, IcNo, AXIS_Z, 200); // 原点検出速度に 200 を設定する(Z軸) [VB.NET] Call Nmc_HomeSpd(No, IcNo, AXIS_Z, 200) [C#] MC8000P.Nmc_HomeSpd(No, IcNo, AXIS.U, 200);</p>
Nmc_LpMax	<p>論理位置カウンタ最大値を設定する。</p> <pre> VC void Nmc_LpMax(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_LpMax(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_LpMax(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 // 論理位置カウンタ最大値に 2000 を設定する(U軸) [VC] Nmc_LpMax(No, IcNo, AXIS_U, 2000); // 論理位置カウンタ最大値に 2000 を設定する(U軸) [VB.NET] Call Nmc_LpMax(No, IcNo, AXIS_U, 2000) [C#] MC8000P.Nmc_LpMax(No, IcNo, AXIS.U, 2000);</p>
Nmc_RpMax	<p>実位置カウンタ最大値を設定する。</p> <pre> VC void Nmc_RpMax(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_RpMax(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_RpMax(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_RpMax(No, IcNo, AXIS_X, 1000); // 実位置カウンタ最大値に 1000 を設定する(X軸) [VB.NET] Call Nmc_RpMax(No, IcNo, AXIS_X, 1000) [C#] MC8000P.Nmc_RpMax(No, IcNo, AXIS.X, 1000);</p>

関数名	機能 及び 内容
Nmc_MR0	<p>多目的レジスタ 0 を設定する。</p> <pre> VC void Nmc_MR0(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_MR0(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_MR0(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_MR0(No, IcNo, AXIS_Z, 500000); // 多目的レジスタ 0 に 500000 を設定する (Z 軸) [VB.NET] Call Nmc_MR0(No, IcNo, AXIS_Z, 500000) [C#] MC8000P.Nmc_MR0(No, IcNo, AXIS.Z, 500000);</p>
Nmc_MR1	<p>多目的レジスタ 1 を設定する。</p> <pre> VC void Nmc_MR1(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_MR1(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_MR1(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_MR1(No, IcNo, AXIS_U, 5000); // 多目的レジスタ 1 に 5000 を設定する (U 軸) [VB.NET] Call Nmc_MR1(No, IcNo, AXIS_U, 5000) [C#] MC8000P.Nmc_MR1(No, IcNo, AXIS.U, 5000);</p>
Nmc_MR2	<p>多目的レジスタ 2 を設定する。</p> <pre> VC void Nmc_MR2(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_MR2(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_MR2(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_MR2(No, IcNo, AXIS_X, 5000000); //多目的レジスタ 2 に 5000000 を設定する (X 軸) [VB.NET] Call Nmc_MR2(No, IcNo, AXIS_X, 5000000) [C#] MC8000P.Nmc_MR2(No, IcNo, AXIS.X, 5000000);</p>

関数名	機能 及び 内容
Nmc_MR3	<p>多目的レジスタ 3 を設定する。</p> <pre> VC void Nmc_MR3(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_MR3(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_MR3(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 Axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_MR3(No, IcNo, AXIS_Y, 30000); //多目的レジスタ 3 に 30000 を設定する(Y軸) [VB.NET] Call Nmc_MR3(No, IcNo, AXIS_Y, 30000) [C#] MC8000P.Nmc_MR3(No, IcNo, AXIS.Y, 30000);</p>
Nmc_SpeedInc	<p>速度増減値を設定する。</p> <pre> VC void Nmc_SpeedInc(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_SpeedInc(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_SpeedInc(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_SpeedInc(No, IcNo, AXIS_Z, 100); // 速度増減値に 100 を設定する(Z軸) [VB.NET] Call Nmc_SpeedInc(No, IcNo, AXIS_Z, 100) [C#] MC8000P.Nmc_SpeedInc(No, IcNo, AXIS.Z, 100);</p>
Nmc_Timer	<p>タイマー値を設定する。</p> <pre> VC void Nmc_Timer(int No, int IcNo, int axis, long wdata); VB.NET Sub Nmc_Timer(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_Timer(int No, int IcNo, AXIS axis, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_Timer(No, IcNo, AXIS_U, 10000); // タイマー値に 10000 を設定する(U軸) [VB.NET] Call Nmc_Timer(No, IcNo, AXIS_U, 10000) [C#] MC8000P.Nmc_Timer(No, IcNo, AXIS.U, 10000);</p>

関数名	機能 及び 内容
Nmc_TPMMax	<p>補間・終点最大値を設定する。</p> <pre> VC void Nmc_TPMMax(int No, int IcNo, long wdata); VB.NET Sub Nmc_TPMMax(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_TPMMax(int No, int IcNo, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_TPMMax(No, IcNo, 10000); // 補間・終点最大値に 10000 を設定する [VB.NET] Call Nmc_TPMMax(No, IcNo, 10000) [C#] MC8000P.Nmc_TPMMax(No, IcNo, 10000);</p>
Nmc_HLNumber	<p>ヘリカル回転数を設定する。</p> <pre> VC void Nmc_HLNumber(int No, int IcNo, long wdata); VB.NET Sub Nmc_HLNumber(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_HLNumber(int No, int IcNo, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_HLNumber(No, IcNo, 7); // ヘリカル回転数に 7 を設定する [VB.NET] Call Nmc_HLNumber(No, IcNo, 7) [C#] MC8000P.Nmc_HLNumber(No, IcNo, 7);</p>
Nmc_HLValue	<p>ヘリカル演算値を設定する。</p> <pre> VC void Nmc_HLValue(int No, int IcNo, long wdata); VB.NET Sub Nmc_HLValue(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_HLValue(int No, int IcNo, int wdata); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_HLValue(No, IcNo, 36799); // ヘリカル演算値に 36799 を設定する [VB.NET] Call Nmc_HLValue(No, IcNo, 36799) [C#] MC8000P.Nmc_HLValue(No, IcNo, 36799);</p>

関数名	機能 及び 内容
Nmc_MRmMode	<p>多目的レジスタモードを設定する。</p> <pre> VC void Nmc_MRmMode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_MRmMode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_MRmMode(int No, int IcNo, AXIS axis, int WR6_data); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 WR6_data 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 // X軸多目的レジスタモードにMR 1の比較対象(現在速度)、比較条件(>)を設定する [VC] Nmc_MRmMode(No, IcNo, AXIS_X, 0x0060); [VB.NET] Call Nmc_MRmMode(No, IcNo, AXIS_X, &H60) [C#] MC8000P.Nmc_MRmMode(No, IcNo, AXIS.X, 0x0060);</p>
Nmc_PI01Mode	<p>P I O信号設定 1 (nPI07~0の機能)を設定する。</p> <pre> VC void Nmc_PI01Mode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_PI01Mode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_PI01Mode(int No, int IcNo, AXIS axis, int WR6_data); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 WR6_data 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_PI01Mode(No, IcNo, AXIS_X, 0x0055); // X軸P I O信号0~3は汎用出力を設定する [VB.NET] Call Nmc_PI01Mode(No, IcNo, AXIS_X, &H55) [C#] MC8000P.Nmc_PI01Mode(No, IcNo, AXIS.X, 0x0055);</p>
Nmc_PI02Mode	<p>P I O信号設定 2・その他(同期パルス出力の論理、パルス幅等)を設定する。</p> <pre> VC void Nmc_PI02Mode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_PI02Mode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_PI02Mode(int No, int IcNo, AXIS axis, int WR6_data); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 WR6_data 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 // X軸パルス信号を正論理、パルス幅を1msecに設定する。 [VC] Nmc_PI02Mode(No, IcNo, AXIS_X, 0x0070); [VB.NET] Call Nmc_PI02Mode(No, IcNo, AXIS_X, &H70) [C#] MC8000P.Nmc_PI02Mode(No, IcNo, AXIS.X, 0x0070);</p>

関数名	機能 及び 内容
Nmc_HMSrch1Mode	<p>自動原点出しモード設定 1 を設定する。</p> <pre> VC void Nmc_HMSrch1Mode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_HMSrch1Mode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_HMSrch1Mode(int No, int IcNo, AXIS axis, int WR6_data); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15))</p> <p>IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照</p> <p>axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。</p> <p>WR6_data 設定するデータ</p> <p>戻り値 なし</p> <p>使用例</p> <p>X 軸自動原点出しモード設定 1 に、以下の設定をする。</p> <ul style="list-style-type: none"> ・ステップ 1, 2, 3, 4 : 実行 ・ステップ1, 2, 3の検出方向 : -方向 ・ステップ1, 2の検出信号 : STOP1 ・ステップ2のDCC出力, RPクリア, LPクリア : 無効 ・ステップ3のDCC出力, RPクリア, LPクリア : 有効 <pre> [VC] Nmc_HMSrch1Mode(No, IcNo, AXIS_X, 0xFC37); [VB.NET] Call Nmc_HMSrch1Mode(No, IcNo, AXIS_X, &HFC37) [C#] MC8000P.Nmc_HMSrch1Mode(No, IcNo, AXIS.X, 0xFC37); </pre>
Nmc_HMSrch2Mode	<p>自動原点出しモード設定 2 を設定する。</p> <pre> VC void Nmc_HMSrch2Mode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_HMSrch2Mode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_HMSrch2Mode(int No, int IcNo, AXIS axis, int WR6_data); </pre> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15))</p> <p>IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照</p> <p>axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。</p> <p>WR6_data 設定するデータ</p> <p>戻り値 なし</p> <p>使用例</p> <p>X 軸自動原点出しモード設定 2 に、以下の設定をする。</p> <ul style="list-style-type: none"> ・ステップ2&3 : 無効 ・原点出し終了時LPクリア : 有効 ・原点出し終了時RPクリア : 有効 ・DCCパルス論理 : 10 μ sec ・DCCパルス幅 : Hiパルス ・ステップ間タイマー : 無効 ・タイマー値 : 0 <pre> [VC] Nmc_HMSrch2Mode(No, IcNo, AXIS_X, 0x0006); [VB.NET] Call Nmc_HMSrch2Mode(No, IcNo, AXIS_X, &H6) [C#] MC8000P.Nmc_HMSrch2Mode(No, IcNo, AXIS.X, 0x0006); </pre>

関数名	機能 及び 内容
Nmc_FilterMode	<p>入力信号フィルタモードを設定する。</p> <pre> VC void Nmc_FilterMode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_FilterMode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_FilterMode(int No, int IcNo, AXIS axis, int WR6_data); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 WR6_data 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 X軸入力信号フィルタモードに、全フィルタ有効、遅延512μsを設定する</p> <pre> [VC] Nmc_FilterMode(No, IcNo, AXIS_X, 0xA AFF); [VB.NET] Call Nmc_FilterMode(No, IcNo, AXIS_X, &H A AFF) [C#] MC8000P.Nmc_FilterMode(No, IcNo, AXIS.X, 0xA AFF); </pre>
Nmc_Sync0Mode	<p>同期動作 S Y N C 0 を設定する。</p> <pre> VC void Nmc_Sync0Mode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_Sync0Mode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_Sync0Mode(int No, int IcNo, AXIS axis, int WR6_data); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 WR6_data 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 「X軸ドライブ中に加減速ドライブの定速域が開始したら、論理位置カウンタをMR0にセーブする」を設定する</p> <pre> [VC] Nmc_Sync0Mode(No, IcNo, AXIS_X, 0x0054); [VB.NET] Call Nmc_Sync0Mode(No, IcNo, AXIS_X, &H54) [C#] MC8000P.Nmc_Sync0Mode(No, IcNo, AXIS.X, 0x0054); </pre>
Nmc_Sync1Mode	<p>同期動作 S Y N C 1 を設定する。</p> <pre> VC void Nmc_Sync1Mode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_Sync1Mode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_Sync1Mode(int No, int IcNo, AXIS axis, int WR6_data); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 WR6_data 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 X軸ドライブ中にMR1対象が真に変化したら、現在タイマー値をMR1にセットする」を設定する</p> <pre> [VC] Nmc_Sync1Mode(No, IcNo, AXIS_X, 0x0071); [VB.NET] Call Nmc_Sync1Mode(No, IcNo, AXIS_X, &H71) [C#] MC8000P.Nmc_Sync1Mode(No, IcNo, AXIS.X, 0x0071); </pre>

関数名	機能 及び 内容
Nmc_Sync2Mode	<p>同期動作 S Y N C 2 を設定する。</p> <p>VC void Nmc_Sync2Mode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_Sync2Mode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_Sync2Mode(int No, int IcNo, AXIS axis, int WR6_data);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 WR6_data 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 「X軸ドライブ中に内蔵タイマーがタイムアップしたら、論理位置カウンタをMR2セーブし、S Y N C 3 を起動する」を設定する</p> <p>[VC] Nmc_Sync2Mode(No, IcNo, AXIS_X, 0x0252); [VB.NET] Call Nmc_Sync2Mode(No, IcNo, AXIS_X, &H252) [C#] MC8000P.Nmc_Sync2Mode(No, IcNo, AXIS.X, 0x0252);</p>
Nmc_Sync3Mode	<p>同期動作 S Y N C 3 を設定する。</p> <p>VC void Nmc_Sync3Mode(int No, int IcNo, int axis, long WR6_data); VB.NET Sub Nmc_Sync3Mode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) C# void MC8000P.Nmc_Sync3Mode(int No, int IcNo, AXIS axis, int WR6_data);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 axis データを設定する軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 WR6_data 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 「X軸ドライブ中にMR3対象が真に変化したら、論理位置カウンタをMR3にセーブし、Y軸 S Y N C 0 を起動する」を設定する</p> <p>[VC] Nmc_Sync3Mode(No, IcNo, AXIS_X, 0x1051); [VB.NET] Call Nmc_Sync3Mode(No, IcNo, AXIS_X, &H1051) [C#] MC8000P.Nmc_Sync3Mode(No, IcNo, AXIS.X, 0x1051);</p>
Nmc_IPMode	<p>補間モードを設定する。</p> <p>VC void Nmc_IPMode(int No, int IcNo, long wdata); VB.NET Sub Nmc_IPMode(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_IPMode(int No, int IcNo, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 wdata 設定するデータ</p> <p>戻り値 なし</p> <p>使用例 [VC] Nmc_IPMode(No, IcNo, 0x0007); //補間軸 X, Y, Z を指定する [VB.NET] Call Nmc_IPMode(No, IcNo, &H7) [C#] MC8000P.Nmc_IPMode(No, IcNo, 0x0007);</p>

関数名	機能 及び 内容
Nmc_ReadLp	<p>論理位置カウンタを読み出す。</p> <pre> VC long Nmc_ReadLp(int No, int IcNo, int axis); VB.NET Function Nmc_ReadLp(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadLp(int No, int IcNo, AXIS axis); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 現在の論理位置カウンタの値</p> <p>使用例 [VC] Data = Nmc_ReadLp(No, IcNo, AXIS_X); // X軸の論理位置カウンタを読み出す [VB.NET] Data = Nmc_ReadLp(No, IcNo, AXIS_X) [C#] Data = MC8000P.Nmc_ReadLp(No, IcNo, AXIS.X);</p>
Nmc_ReadEp	<p>実位置カウンタを読み出す。</p> <pre> VC long Nmc_ReadEp(int No, int IcNo, int axis); VB.NET Function Nmc_ReadEp(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadEp(int No, int IcNo, AXIS axis); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 現在の実位置カウンタの値</p> <p>使用例 [VC] Data = Nmc_ReadEp(No, IcNo, AXIS_Y); // Y軸の実位置カウンタを読み出す [VB.NET] Data = Nmc_ReadEp(No, IcNo, AXIS_Y) ' Y軸の実位置カウンタを読み出す [C#] Data = MC8000P.Nmc_ReadEp(No, IcNo, AXIS.Y)</p>
Nmc_ReadSpeed	<p>現在ドライブ速度を読み出す。</p> <pre> VC long Nmc_ReadSpeed(int No, int IcNo, int axis); VB.NET Function Nmc_ReadSpeed(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadSpeed(int No, int IcNo, AXIS axis); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 現在ドライブ速度の値</p> <p>使用例 [VC] Data = Nmc_ReadSpeed(No, IcNo, AXIS_Z); // Z軸の現在ドライブ速度を読み出す [VB.NET] Data = Nmc_ReadSpeed(No, IcNo, AXIS_Z) [C#] Data = MC8000P.Nmc_ReadSpeed(No, IcNo, AXIS.Z);</p>

関数名	機能 及び 内容
Nmc_ReadAccDec	<p>現在加／減速度を読み出す。</p> <p>ドライブ中の現在加速度、または減速度の値を読み出す。 ドライブ停止時の読み出しデータは不定です。</p> <p>VC long Nmc_ReadAccDec(int No, int IcNo, int axis); VB.NET Function Nmc_ReadAccDec(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadAccDec(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 現在加／減速度の値</p> <p>使用例 [VC] Data = Nmc_ReadAccDec(No, IcNo, AXIS_U); // U軸の現在加／減速度を読み出す [VB.NET] Data = Nmc_ReadAccDec(No, IcNo, AXIS_U) [C#] Data = MC8000P.Nmc_ReadAccDec(No, IcNo, AXIS.U);</p>
Nmc_ReadMRO	<p>多目的レジスタ0を読み出す。</p> <p>VC long Nmc_ReadMRO(int No, int IcNo, int axis); VB.NET Function Nmc_ReadMRO(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadMRO(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 多目的レジスタ0の値</p> <p>使用例 [VC] Data = Nmc_ReadMRO(No, IcNo, AXIS_X); // X軸の多目的レジスタ0を読み出す [VB.NET] Data = Nmc_ReadMRO(No, IcNo, AXIS_X) [C#] Data = MC8000P.Nmc_ReadMRO(No, IcNo, AXIS.X);</p>
Nmc_ReadMR1	<p>多目的レジスタ1を読み出す。</p> <p>VC long Nmc_ReadMR1(int No, int IcNo, int axis); VB.NET Function Nmc_ReadMR1(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadMR1(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 多目的レジスタ1の値</p> <p>使用例 [VC] Data = Nmc_ReadMR1(No, IcNo, AXIS_Y); // Y軸の多目的レジスタ1を読み出す [VB.NET] Data = Nmc_ReadMR1(No, IcNo, AXIS_Y) [C#] Data = MC8000P.Nmc_ReadMR1(No, IcNo, AXIS.Y);</p>

関数名	機能 及び 内容
Nmc_ReadMR2	<p>多目的レジスタ 2 を読み出す。</p> <p>VC long Nmc_ReadMR2(int No, int IcNo, int axis); VB.NET Function Nmc_ReadMR2(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadMR2(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 多目的レジスタ 2 の値</p> <p>使用例 [VC] Data = Nmc_ReadMR2(No, IcNo, AXIS_Z); // Z軸の多目的レジスタ 2 を読み出す [VB.NET] Data = Nmc_ReadMR2(No, IcNo, AXIS_Z) [C#] Data = MC8000P.Nmc_ReadMR2(No, IcNo, AXIS.Z);</p>
Nmc_ReadMR3	<p>多目的レジスタ 3 を読み出す。</p> <p>VC long Nmc_ReadMR3(int No, int IcNo, int axis); VB.NET Function Nmc_ReadMR3(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadMR3(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 多目的レジスタ 3 の値</p> <p>使用例 [VC] Data = Nmc_ReadMR3(No, IcNo, AXIS_U); // U軸の多目的レジスタ 3 を読み出す [VB.NET] Data = Nmc_ReadMR3(No, IcNo, AXIS_U) [C#] Data = MC8000P.Nmc_ReadMR3(No, IcNo, AXIS.U);</p>
Nmc_ReadCT	<p>現在タイマー値を読み出す。</p> <p>VC long Nmc_ReadCT(int No, int IcNo, int axis); VB.NET Function Nmc_ReadCT(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadCT(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 現在タイマー値の値</p> <p>使用例 [VC] Data = Nmc_ReadCT(No, IcNo, AXIS_X); // X軸の現在タイマー値を読み出す [VB.NET] Data = Nmc_ReadCT(No, IcNo, AXIS_X) [C#] Data = MC8000P.Nmc_ReadMR3(No, IcNo, AXIS.X);</p>

関数名	機能 及び 内容
Nmc_ReadTX	<p>補間・終点最大値を読み出す。</p> <pre> VC long Nmc_ReadTX(int No, int IcNo); VB.NET Function Nmc_ReadTX(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadTX(int No, int IcNo); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 補間・終点最大値の値</p> <p>使用例 [VC] Data = Nmc_ReadTX(No, IcNo); // 補間・終点最大値を読み出す [VB.NET] Data = Nmc_ReadTX(No, IcNo) [C#] Data = MC8000P.Nmc_ReadTX(No, IcNo);</p> <p>注意 補間ドライブの実行前と、実行中では読み出される値が異なります。補間ドライブ実行前は、入力中の補間セグメントの最終最大値が読み出されます。補間ドライブ実行中は現在実行中の補間セグメントの終点最大値が読み出されます。</p>
Nmc_ReadCHLN	<p>現在ヘリカル回転数を読み出す</p> <pre> VC long Nmc_ReadCHLN(int No, int IcNo); VB.NET Function Nmc_ReadCHLN(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadCHLN(int No, int IcNo); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 現在ヘリカル回転数の値</p> <p>使用例 [VC] Data = Nmc_ReadCHLN(No, IcNo); // 現在ヘリカル回転数を読み出す [VB.NET] Data = Nmc_ReadCHLN(No, IcNo) [C#] Data = MC8000P.Nmc_ReadCHLN(No, IcNo);</p>
Nmc_ReadHLV	<p>ヘリカル演算値を読み出す。 ヘリカル演算命令(6Bh, 6Ch)でヘリカル演算を行った結果を読み出すときに使用します。</p> <pre> VC long Nmc_ReadHLV(int No, int IcNo); VB.NET Function Nmc_ReadHLV(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_ReadHLV(int No, int IcNo); </pre> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 ヘリカル演算値の値</p> <p>使用例 [VC] Data = Nmc_ReadHLV(No, IcNo); // ヘリカル演算値を読み出す [VB.NET] Data = Nmc_ReadHLV(No, IcNo) [C#] Data = MC8000P.Nmc_ReadHLV(No, IcNo);</p>

関数名	機能 及び 内容
Nmc_ReadWR1	<p>WR 1 設定値を読み出す。</p> <p>VC long Nmc_ReadWR1(int No, int IcNo, int axis); VB.NET Function Nmc_ReadWR1(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadWR1(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 WR 1 の値</p> <p>使用例 [VC] Data = Nmc_ReadWR1(No, IcNo, AXIS_Y); // Y軸のWR 1 設定値を読み出す [VB.NET] Data = Nmc_ReadWR1(No, IcNo, AXIS_Y) [C#] Data = MC8000P.Nmc_ReadWR1(No, IcNo, AXIS.Y);</p>
Nmc_ReadWR2	<p>WR 2 設定値を読み出す。</p> <p>VC long Nmc_ReadWR2(int No, int IcNo, int axis); VB.NET Function Nmc_ReadWR2(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadWR2(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 WR 2 の値</p> <p>使用例 [VC] Data = Nmc_ReadWR2(No, IcNo, AXIS_Z); // Z軸のWR 2 設定値を読み出す [VB.NET] Data = Nmc_ReadWR2(No, IcNo, AXIS_Z) ' Z軸のWR 2 設定値を読み出す [C#] Data = MC8000P.Nmc_ReadWR2(No, IcNo, AXIS.Z);</p>
Nmc_ReadWR3	<p>WR 3 設定値を読み出す。</p> <p>VC long Nmc_ReadWR3(int No, int IcNo, int axis); VB.NET Function Nmc_ReadWR3(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadWR3(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 WR 3 の値</p> <p>使用例 [VC] Data = Nmc_ReadWR3(No, IcNo, AXIS_U); // U軸のWR 3 設定値を読み出す [VB.NET] Data = Nmc_ReadWR3(No, IcNo, AXIS_U) [C#] Data = MC8000P.Nmc_ReadWR3(No, IcNo, AXIS.U);</p>

関数名	機能 及び 内容
Nmc_ReadMRM	<p>多目的レジスタモード設定を読み出す。</p> <p>VC long Nmc_ReadMRM(int No, int IcNo, int axis); VB.NET Function Nmc_ReadMRM(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadMRM(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 多目的レジスタモードの値</p> <p>使用例 [VC] Data = Nmc_ReadMRM(No, IcNo, AXIS_X); // X軸の多目的レジスタモード設定値を読み出す [VB.NET] Data = Nmc_ReadMRM(No, IcNo, AXIS_X) [C#] Data = MC8000P.Nmc_ReadMRM(No, IcNo, AXIS.X);</p>
Nmc_ReadP1M	<p>P I O信号設定 1 を読み出す。</p> <p>VC long Nmc_ReadP1M(int No, int IcNo, int axis); VB.NET Function Nmc_ReadP1M(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadP1M(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 P I O信号設定 1 の値</p> <p>使用例 [VC] Data = Nmc_ReadP1M(No, IcNo, AXIS_Y); // Y軸のP I O信号設定 1 設定値を読み出す [VB.NET] Data = Nmc_ReadP1M(No, IcNo, AXIS_Y) [C#] Data = MC8000P.Nmc_ReadP1M(No, IcNo, AXIS.Y);</p>
Nmc_ReadP2M	<p>P I O信号設定 2 ・その他設定を読み出す。</p> <p>VC long Nmc_ReadP2M(int No, int IcNo, int axis); VB.NET Function Nmc_ReadP2M(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadP2M(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 P I O信号設定 2 ・その他設定の値</p> <p>使用例 // Z軸のP I O信号設定 2 ・その他設定値を読み出す [VC] Data = Nmc_ReadP2M(No, IcNo, AXIS_Z); [VB.NET] Data = Nmc_ReadP2M(No, IcNo, AXIS_Z) [C#] Data = MC8000P.Nmc_ReadP2M(No, IcNo, AXIS.Z);</p>

関数名	機能 及び 内容
Nmc_ReadAc	<p>加速度設定値を読み出す。</p> <p>VC long Nmc_ReadAc(int No, int IcNo, int axis); VB.NET Function Nmc_ReadAc(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadAc(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 設定した加速度の値</p> <p>使用例 [VC] Data = Nmc_ReadAc(No, IcNo, AXIS_U); // U軸の加速度設定値を読み出す [VB.NET] Data = Nmc_ReadAc(No, IcNo, AXIS_U) [C#] Data = MC8000P.Nmc_ReadAc(No, IcNo, AXIS.U);</p>
Nmc_ReadStartSpd	<p>初速度設定値を読み出す。</p> <p>VC long Nmc_ReadStartSpd(int No, int IcNo, int axis); VB.NET Function Nmc_ReadStartSpd(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadStartSpd(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 設定した初速度の値</p> <p>使用例 [VC] Data = Nmc_ReadStartSpd(No, IcNo, AXIS_X); // X軸の初速度設定値を読み出す [VB.NET] Data = Nmc_ReadStartSpd(No, IcNo, AXIS_X) [C#] Data = MC8000P.Nmc_ReadStartSpd(No, IcNo, AXIS.X);</p>
Nmc_ReadSetSpeed	<p>ドライブ速度設定値を読み出す。</p> <p>VC long Nmc_ReadSetSpeed(int No, int IcNo, int axis); VB.NET Function Nmc_ReadSetSpeed(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadSetSpeed(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 設定したドライブ速度の値</p> <p>使用例 [VC] Data = Nmc_ReadSetSpeed(No, IcNo, AXIS_Y); // Y軸のドライブ速度設定値を読み出す [VB.NET] Data = Nmc_ReadSetSpeed(No, IcNo, AXIS_Y) [C#] Data = MC8000P.Nmc_ReadSetSpeed(No, IcNo, AXIS.Y);</p>

関数名	機能 及び 内容
Nmc_ReadPulse	<p>移動パルス数/終点設定値を読み出す。</p> <p>VC long Nmc_ReadPulse(int No, int IcNo, int axis); VB.NET Function Nmc_ReadPulse(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_ReadPulse(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 設定した移動パルス数/終点設定値の値</p> <p>使用例 [VC] Data = Nmc_ReadPulse(No, IcNo, AXIS_Z); // Z軸の移動パルス数/終点設定値を読み出す [VB.NET] Data = Nmc_ReadPulse(No, IcNo, AXIS_Z) [C#] Data = MC8000P.Nmc_ReadSetSpeed(No, IcNo, AXIS.Z);</p>
Nmc_GetDriveStatus	<p>ドライブ状態を取得する。指定軸のドライブが終了したか調べる時に使用する。</p> <p>VC int Nmc_GetDriveStatus(int No, int IcNo, int axis); VB.NET Function Nmc_GetDriveStatus(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer C# int MC8000P.Nmc_GetDriveStatus(int No, int IcNo, AXIS axis);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis ドライブ状態を取得する軸。複数軸指定可能。 詳細は「5.1.3 補足説明」(2)参照。</p> <p>戻り値 指定した全ての軸のドライブが終了している場合は0を返す。 指定した軸のうち1つ以上がドライブ中の場合は、0以外を返す。</p> <p>使用例 [VC] if(Nmc_GetDriveStatus(No, IcNo, AXIS_X) == 0) // X軸がドライブ終了の場合 AfxMessageBox("X軸ドライブ終了"); else AfxMessageBox("X軸ドライブ中"); [VB.NET] If Nmc_GetDriveStatus(No, IcNo, AXIS_X) = 0 Then Call MsgBox("X軸ドライブ終了") Else Call MsgBox("X軸ドライブ中") End If [C#] if(MC8000P.Nmc_GetDriveStatus(No, IcNo, AXIS.X) == 0) MessageBox.Show("X軸ドライブ終了"); else MessageBox.Show("X軸ドライブ中");</p>

関数名	機能 及び 内容
Nmc_GetCNextStatus	<p>連続補間次データ書き込み可能状態を取得する。 連続補間実行中に次データ書き込み可能になったか調べる時に使用する。</p> <p>VC int Nmc_GetCNextStatus(int No, int IcNo); VB. NET Function Nmc_GetCNextStatus(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_GetCNextStatus(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 連続補間次データ書き込み可能の場合は、0以外を返す。 連続補間次データ書き込み可能ではない場合は、0を返す。</p> <p>使用例 [VC] if(Nmc_GetCNextStatus(No, IcNo) != 0) // 次データ書き込み可能の場合 AfxMessageBox("連続補間次データ書き込み可能である"); else AfxMessageBox("連続補間次データ書き込み可能ではない"); [VB. NET] If Nmc_GetCNextStatus(No, IcNo) <> 0 Then Call MsgBox("連続補間次データ書き込み可能である") Else Call MsgBox("連続補間次データ書き込み可能ではない") End If [C#] if(MC8000P.Nmc_GetCNextStatus(No, IcNo) != 0) MessageBox.Show("連続補間次データ書き込み可能である"); else MessageBox.Show("連続補間次データ書き込み可能ではない");</p>
Nmc_GetSc	<p>連続補間プリバッファスタックカウンタの値を取得する。</p> <p>VC int Nmc_GetSc(int No, int IcNo); VB. NET Function Nmc_GetSc(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# int MC8000P.Nmc_GetSc(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 現在のプリバッファスタックカウンタの値</p> <p>使用例 [VC] Data = Nmc_GetSc(No, IcNo); // プリバッファスタックカウンタの値を取得 [VB. NET] Data = Nmc_GetSc(No, IcNo) [C#] Data = MC8000P.Nmc_GetSc(No, IcNo)</p>

関数名	機能 及び 内容
Nmc_WriteRegSetAxis	<p>指定軸の指定したライトレジスタ (WR 1 ~ WR 3 のいずれか) にデータを書き込む。</p> <p>VC void Nmc_WriteRegSetAxis(int No, int IcNo, int axis, int RegNumber, long wdata); VB.NET Sub Nmc_WriteRegSetAxis(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal RegNumber As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteRegSetAxis(int No, int IcNo, AXIS axis, int RegNumber, int wdata);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 axis データを書き込む軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 RegNumber データを書き込むライトレジスタの番号 [VC] [VB.NET] WR1はMCX_WR1, WR2はMCX_WR2, WR3はMCX_WR3 を指定する。 [C#] RR1はREG_MCX.RR1, RR2はREG_MCX.RR2 を指定する。 詳細は「5.1.3 補足説明」(1)参照。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 ハードリミット有効を設定する [VC] Nmc_WriteRegSetAxis(No, IcNo, AXIS_ALL, MCX_WR2, 0x0800); [VB.NET] Call Nmc_WriteRegSetAxis(No, IcNo, AXIS_ALL, MCX_WR2, &H800) [C#] MC8000P.Nmc_WriteRegSetAxis(No, IcNo, AXIS.ALL, REG_MCX.WR2, 0x0800);</p>
Nmc_ReadRegSetAxis	<p>指定軸の指定したリードレジスタ (RR 2、RR 3 のどちらか) のデータを読み出す。</p> <p>VC long Nmc_ReadRegSetAxis(int No, int IcNo, int axis, int RegNumber); VB.NET Function Nmc_ReadRegSetAxis(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal RegNumber As Integer) As Integer C# int MC8000P.Nmc_ReadRegSetAxis(int No, int IcNo, AXIS axis, int RegNumber);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 axis データを読み出す軸。 X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。 RegNumber データを読み出すリードレジスタの番号 [VC] [VB.NET] RR1はMCX_RR1, RR2はMCX_RR2 を指定する。 [C#] RR1はREG_MCX.RR1, RR2はREG_MCX.RR2 を指定する。 詳細は「5.1.3 補足説明」(1)参照。</p> <p>戻り値 指定軸の指定したリードレジスタのデータ</p> <p>使用例 X軸のRR2のデータを読み出す [VC] Data = Nmc_ReadRegSetAxis(No, IcNo, AXIS_X, MCX_RR2); [VB.NET] Data = Nmc_ReadRegSetAxis(No, IcNo, AXIS_X, MCX_RR2) [C#] Data = MC8000P.Nmc_ReadRegSetAxis(No, IcNo, AXIS.X, REG_MCX.RR2);</p> <p>注意 RR3 (ステータスレジスタ3) は、ページ0およびページ1の2種類存在します。読み出す前にRR3 ページ表示命令 (7Ah、7Bh) を書き込むことでページの指定をします。リセット時はページ0となります。</p>

関数名	機能 及び 内容
Nmc_WriteData	<p>指定軸に指定したパラメータのデータを書き込む。(データ書き込み命令を実行する)</p> <p>VC void Nmc_WriteData(int No, int IcNo, int axis, int cmd, long wdata); VB.NET Sub Nmc_WriteData(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal cmd As Integer, ByVal wdata As Integer) C# void MC8000P.Nmc_WriteData(int No, int IcNo, AXIS axis, CMD cmd, int wdata);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを書き込む軸。複数軸指定可能。詳細は「5.1.3 補足説明」(2)参照。 cmd 書き込み命令コード((00)H~(16)H, (19)H~(1B)H)。例: 加速度増加率設定は(00)H。 wdata 書き込むデータ</p> <p>戻り値 なし</p> <p>使用例 全軸のドライブ速度に1000を設定する。ドライブ速度の命令コードは(05)H [VC] Nmc_WriteData(No, IcNo, AXIS_ALL, 0x05, 1000); [VB.NET] Call Nmc_WriteData(No, IcNo, AXIS_ALL, &H5, 1000) [C#] MC8000P.Nmc_WriteData(No, IcNo, AXIS.ALL, 0x05, 1000);</p>
Nmc_ReadData	<p>データ読み出し命令を実行し、データを読み出す。</p> <p>VC long Nmc_ReadData(int No, int IcNo, int axis, int cmd); VB.NET Function Nmc_ReadData(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal cmd As Integer) As Integer C# int MC8000P.Nmc_ReadData(int No, int IcNo, AXIS axis, CMD cmd);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。 axis データを読み出す軸。X軸はAXIS_X, Y軸はAXIS_Y, Z軸はAXIS_Z, U軸はAXIS_Uを指定する。 詳細は「5.1.3 補足説明」(2)参照。 cmd データ読み出し命令コード((30)H~(46)H)。例: 論理位置カウンタ読み出しは(30)H。</p> <p>戻り値 読み出したデータ</p> <p>使用例 X軸の論理位置カウンタを読み出す。 [VC] Data = Nmc_ReadData(No, IcNo, AXIS_X, 0x30); [VB.NET] Data = Nmc_ReadData(No, IcNo, AXIS_X, &H30) [C#] Data = MC8000P.Nmc_ReadData(No, IcNo, AXIS.X, 0x30);</p>


```

Data2Bp(1).Bp2p = &HFS
Data2Bp(1).Bp2m = &H3FC0S

Call Nmc_IPMode(No, IcNo, &H3) ' 補間モード設定。X, Y 軸指定。

' 速度関連パラメータ設定(実際の設定記述は省略)

Ret = Nmc_2BPExecMC8500P(No, IcNo, Data2Bp, 2, &H3) ' 2軸BP補間実行。データ数2, X, Y軸

If Ret = BP_END Then ' 戻り値正常
  Call MsgBox("正常終了")
End If

```

[C#]

```

DATA_2BP [] Data2Bp = new DATA_2BP[2]; // 2軸BP補間データ
// 補間データ設定
Data2Bp[0].Bp1p = 0; // 0000 0000 0000 0000 BP1+方向 0パルス
Data2Bp[0].Bp1m = 0x2BFF; // 0010 1011 1111 1111 BP1-方向 12パルス
Data2Bp[0].Bp2p = 0xFFD4; // 1111 1111 1101 0100 BP2+方向 12パルス
Data2Bp[0].Bp2m = 0; // 0000 0000 0000 0000 BP2-方向 0パルス

Data2Bp[1].Bp1p = 0xF6FE; // 1111 0110 1111 1110 BP1+方向 13パルス
Data2Bp[1].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data2Bp[1].Bp2p = 0xF; // 0000 0000 0000 1111 BP2+方向 4パルス
Data2Bp[1].Bp2m = 0x3FC0; // 0011 1111 1100 0000 BP2-方向 8パルス

MC8000P.Nmc_IPMode(No, IcNo, 0x0003); //補間モード設定。X, Y 軸指定。

// 速度関連パラメータ設定(実際の設定記述は省略)

// 2軸BP補間実行。データ数2, X, Y軸

Ret = MC8000.Nmc_2BPExecMC8500P(No, IcNo, Data2Bp, 2, 0x3);

if(Ret == Nmc_Status.BP_END) // 戻り値正常

```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸などを設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定してください。


```

Data3Bp(0).Bp3m = &HAC35S

Data3Bp(1).Bp1p = &HAC35S
Data3Bp(1).Bp1m = 0
Data3Bp(1).Bp2p = &HC000S
Data3Bp(1).Bp2m = &H36E7S
Data3Bp(1).Bp3p = &HC000S
Data3Bp(1).Bp3m = &H3F3FS

Call Nmc_IPMode(No, IcNo, &H7) ' 補間モード設定。X, Y, Z 軸指定。

' 速度関連パラメータ設定 (実際の設定記述は省略)

Ret = Nmc_3BPExecMC8500P(No, IcNo, Data3Bp, 2, &H7) ' 3 軸 B P 補間実行。データ数2, X, Y, Z 軸

If Ret = BP_END Then ' 戻り値正常
    Call MsgBox("正常終了")
End If

```

[C#]

```

DATA_3BP [] Data3Bp = new DATA_3BP[2]; // 3 軸 B P 補間データ
// 補間データ設定
Data3Bp[0].Bp1p = 0xFF30; // 1111 1111 0011 0000 BP1+方向 10パルス
Data3Bp[0].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data3Bp[0].Bp2p = 0; // 0000 0000 0000 0000 BP2+方向 0パルス
Data3Bp[0].Bp2m = 0x84FF; // 1000 0100 1111 1111 BP2-方向 10パルス
Data3Bp[0].Bp3p = 0; // 0000 0000 0000 0000 BP3+方向 0パルス
Data3Bp[0].Bp3m = 0xAC35; // 1010 1100 0011 0101 BP3-方向 8パルス

Data3Bp[1].Bp1p = 0xAC35; // 1010 1100 0011 0101 BP1+方向 8パルス
Data3Bp[1].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data3Bp[1].Bp2p = 0xC000; // 1100 0000 0000 0000 BP2+方向 2パルス
Data3Bp[1].Bp2m = 0x36E7; // 0011 0110 1110 0111 BP2-方向 10パルス
Data3Bp[1].Bp3p = 0xC000; // 1100 0000 0000 0000 BP3+方向 2パルス
Data3Bp[1].Bp3m = 0x3F3F; // 0011 1111 0011 1111 BP3-方向 12パルス

MC8000P.Nmc_IPMode(No, IcNo, 0x0007); // 補間モード設定。X, Y, Z 軸指定。

// 速度関連パラメータ設定 (実際の設定記述は省略)

Ret = MC8000P.Nmc_3BPExecMC8500P(No, IcNo, Data3Bp, 2, 0x7); // 3 軸 B P 補間実行

if(Ret == Nmc_Status.BP_END) // 戻り値正常

```

注意

この関数を実行する前に、必ず補間モード設定 (2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定してください。

関数名	機能 及び 内容
Nmc_4BPExecMC8500P	<p>指定した補間データで4軸ビットパターン補間を実行する。 この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。</p> <p>VC DWORD Nmc_4BPExecMC8500P(int No, int IcNo, DATA_4BP* pData4Bp, int DataCnt, int IpAxis);</p> <p>VB.NET Function Nmc_4BPExecMC8500P(ByVal No As Integer, ByVal IcNo As Integer, ByVal pData4Bp As DATA_4BP(), ByVal DataCnt As Integer, ByVal IpAxis As Integer) As Integer</p> <p>C# Nmc_Status MC8000P.Nmc_4BPExecMC8500P(int No, int IcNo, DATA_4BP[] pData4Bp, int DataCnt, int IpAxis);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 pData4Bp 4軸BP補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_4BPのアドレス)。 DATA_4BPに実行する補間データをセットし、アドレスを指定する。 DATA_4BPについては「5.1.3 補足説明」(3)参照。 DataCnt 4軸BP補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis 補間を実行する軸。補間モード設定(2A)HのD0~D3(軸指定)の設定値と同じ値を指定する。 「5.1.3 補足説明」(4)参照。</p> <p>戻り値</p> <p>補間処理が正常終了した場合は BP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は「5.1.3 補足説明」(1)参照。</p> <p>■正常終了 BP_END BP補間処理正常終了</p> <p>■エラーコード</p> <p>BP_CNT_ERR 指定されたデータ数が範囲外 BP_ALREADY_EXEC 既にBP補間、あるいは連続補間が実行中 BP_MALLOC_ERR メモリを確保できなかった BP_PARAM_ERR 引数の値が正しくない BP_NOT_OPEN_ERR 指定したボードがオープンされていない BP_OTHER_ERR その他のエラー BP_FUNC_ERR 使用できない関数を使用(MC8500PでMC8000P用関数を使用したときなど) BP_STOP BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった場合) BP_USER_STOP BP補間実行中にユーザーが中断した BP_DRIVE_ERR BP補間実行中にボードでエラー発生(RR0にエラー情報がセットされた) BP_STOP_CERR 連続補間が途中で停止した(RR2エラーによる停止の場合) BP_GETSC_ERR スタックカウンタ読み出しエラー</p> <p>補間データに終了コードが書かれていて補間が終了した場合は、BP_ENDまたはBP_STOPのいずれかのエラーコードが返ります。</p> <p>使用例</p> <p>[VC]</p> <pre>// 4軸BP補間データ BP1P, BP1M, BP2P, BP2M, BP3P, BP3M, BP4P, BP4M DATA_4BP Data4Bp[2] = { {0xFFE4, 0x0000, 0x03FF, 0x4000, 0x0000, 0xFFFF, 0xFE80, 0x000F}, {0x0000, 0x03FF, 0xFFD0, 0x0000, 0x4AAB, 0x0000, 0x1FFF, 0x0000} }; Nmc_IPMode(No, IcNo, 0x000F); //補間モード設定。X, Y, Z, U軸指定。 // 速度関連パラメータ設定(実際の設定記述は省略) // 4軸BP補間実行。データ数2, X, Y, Z, U軸 Ret = Nmc_4BPExecMC8500P(No, IcNo, Data4Bp, 2, 0xF); if(Ret == BP_END) AfxMessageBox("正常終了"); //戻り値正常</pre> <p>[VB.NET]</p> <pre>' 4軸BP補間データ Dim Data4Bp(1) As DATA_4BP Data4Bp(0).Bp1p = &HFFE4S Data4Bp(0).Bp1m = 0 Data4Bp(0).Bp2p = &H3FFS Data4Bp(0).Bp2m = &H4000S</pre>

```
Data4Bp(0).Bp3p = 0
Data4Bp(0).Bp3m = &HFFFFS
Data4Bp(0).Bp4p = &HFE80S
Data4Bp(0).Bp4m = &H000FS
```

```
Data4Bp(1).Bp1p = 0
Data4Bp(1).Bp1m = &H3FFS
Data4Bp(1).Bp2p = &HFFD0S
Data4Bp(1).Bp2m = 0
Data4Bp(1).Bp3p = &H4AABS
Data4Bp(1).Bp3m = 0
Data4Bp(1).Bp4p = &H1FFFS
Data4Bp(1).Bp4m = 0
```

```
Call Nmc_IPMode(No, IcNo, &HF) ' 補間モード設定。X, Y, Z, U軸指定。
```

```
' 速度関連パラメータ設定(実際の設定記述は省略)
```

```
' 4軸BP補間実行。データ数2, X, Y, Z, U軸
Ret = Nmc_4BPExecMC8500P(No, IcNo, Data4Bp, 2, &HF)
```

```
If Ret = BP_END Then
  Call MsgBox("正常終了")
End If
```

[C#]

```
DATA_4BP [] Data4Bp = new DATA_4BP[2]; // 4軸BP補間データ
// 補間データ設定
Data4Bp[0].Bp1p = 0xFFE4; // 1111 1111 1110 0100 BP1+方向 12パルス
Data4Bp[0].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data4Bp[0].Bp2p = 0x03FF; // 0000 0011 1111 1111 BP2+方向 10パルス
Data4Bp[0].Bp2m = 0x4000; // 0100 0000 0000 0000 BP2-方向 1パルス
Data4Bp[0].Bp3p = 0; // 0000 0000 0000 0000 BP3+方向 0パルス
Data4Bp[0].Bp3m = 0xFFFF; // 1111 1111 1111 1111 BP3-方向 16パルス
Data4Bp[0].Bp4p = 0xFE80; // 1111 1110 1000 0000 BP4+方向 8パルス
Data4Bp[0].Bp4m = 0x000F; // 0000 0000 0000 1111 BP4-方向 4パルス

Data4Bp[1].Bp1p = 0; // 0000 0000 0000 0000 BP1+方向 0パルス
Data4Bp[1].Bp1m = 0x03FF; // 0000 0011 1111 1111 BP1-方向 10パルス
Data4Bp[1].Bp2p = 0xFFD0; // 1111 1111 1101 0000 BP2+方向 11パルス
Data4Bp[1].Bp2m = 0; // 0000 0000 0000 0000 BP2-方向 0パルス
Data4Bp[1].Bp3p = 0x4AAB; // 0100 1010 1010 1011 BP3+方向 8パルス
Data4Bp[1].Bp3m = 0; // 0000 0000 0000 0000 BP3-方向 0パルス
Data4Bp[1].Bp4p = 0x1FFF; // 0001 1111 1111 1111 BP4+方向 13パルス
Data4Bp[1].Bp4m = 0; // 0000 0000 0000 0000 BP4-方向 0パルス
```

```
MC8000P.Nmc_IPMode(No, IcNo, 0x000F); //補間モード設定。X, Y, Z, U軸指定。
```

```
// 速度関連パラメータ設定(実際の設定記述は省略)
```

```
Ret = MC8000P.Nmc_4BPExecMC8500P(No, IcNo, Data4Bp, 1, 0xF); // 4軸BP補間実行
```

```
if(Ret == Nmc_Status.BP_END) // 戻り値正常
```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定してください。


```

// 速度関連パラメータ設定(実際の設定記述は省略)

Ret = Nmc_2BPExecMC8500P_BG(hWnd, No, IcNo, Data2Bp, 2, 0x3);
//2軸BP補間実行。データ数2, X, Y軸

if(Ret == BP_START) AfxMessageBox("補間開始"); //戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog) // WM_BP_ENDメッセージ受信関数設定
ON_MESSAGE(WM_BP_END, OnMsg_BP)
END_MESSAGE_MAP()

// WM_BP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_BP(WPARAM BoardNo, LPARAM Status)
{
if(Status == BP_END) AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
return 0;
}

[VB.NET]
' 2軸BP補間データ
Dim Data2Bp(1) As DATA_2BP
Data2Bp(0).Bp1p = 0
Data2Bp(0).Bp1m = &H2BFFS
Data2Bp(0).Bp2p = &HFFD4S
Data2Bp(0).Bp2m = 0

Data2Bp(1).Bp1p = &HF6FES
Data2Bp(1).Bp1m = 0
Data2Bp(1).Bp2p = &HFS
Data2Bp(1).Bp2m = &H3FC0S
Call Nmc_IPMode(No, IcNo, &H3) '補間モード設定。X, Y軸指定。

' 速度関連パラメータ設定(実際の設定記述は省略)

' 2軸BP補間実行。データ数2, X, Y軸
Ret = Nmc_2BPExecMC8500P_BG(Handle.ToInt32, No, IcNo, Data2Bp, 2, &H3)
If Ret = BP_START Then ' 戻り値正常(補間開始)
Call MsgBox("補間開始")
End If
End Sub

' WM_BP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)
Select Case (m.Msg)
Case WM_BP_END ' BP補間終了メッセージ
If m.LParam.ToInt32 = BP_END Then ' 戻り値正常(補間終了)
Call MsgBox("補間正常終了")
End If
Exit Select
End Select
MyBase.WndProc(m)
End Sub

[C#]
DATA_2BP [] Data2Bp = new DATA_2BP[2]; // 2軸BP補間データ
// 補間データ設定
Data2Bp[0].Bp1p = 0; // 0000 0000 0000 0000 BP1+方向 0パルス
Data2Bp[0].Bp1m = 0x2BFF; // 0010 1011 1111 1111 BP1-方向 12パルス
Data2Bp[0].Bp2p = 0xFFD4; // 1111 1111 1101 0100 BP2+方向 12パルス
Data2Bp[0].Bp2m = 0; // 0000 0000 0000 0000 BP2-方向 0パルス

Data2Bp[1].Bp1p = 0xF6FE; // 1111 0110 1111 1110 BP1+方向 13パルス
Data2Bp[1].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data2Bp[1].Bp2p = 0xF; // 0000 0000 0000 1111 BP2+方向 4パルス
Data2Bp[1].Bp2m = 0x3FC0; // 0011 1111 1100 0000 BP2-方向 8パルス

MC8000P.Nmc_IPMode(No, IcNo, 0x0003); //補間モード設定。X, Y軸指定。

// 速度関連パラメータ設定(実際の設定記述は省略)

```

```

// 2軸BP補間バックグラウンド実行
Ret = MC8000P.Nmc_2BPExecMC8500P_BG((System.IntPtr)parent.Handle, No, IcNo, Data2Bp, 2, 0x3);
if(Ret == Nmc_Status.BP_START) // 補間開始正常の場合

//WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    // 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
    base.WndProc ( ref m );
    if ( m.Msg == (int)MSG_ID.WM_BP_END )
    {
        if((uint)m.LParam == Nmc_Status.BP_END) // BP補間処理正常終了
    }
}

```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定して下さい。


```

// 速度関連パラメータ設定(実際の設定記述は省略)
Ret = Nmc_3BPExecMC8500P_BG(hWnd, No, IcNo, Data3Bp, 2, 0x7);
//3軸BP補間実行。データ数2, X, Y, Z軸
if(Ret == BP_START) AfxMessageBox("補間開始"); //戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog) // WM_BP_ENDメッセージ受信関数設定
ON_MESSAGE(WM_BP_END, OnMsg_BP)
END_MESSAGE_MAP()

// WM_BP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_BP(WPARAM BoardNo, LPARAM Status)
{
if(Status == BP_END) AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
return 0;
}

[VB.NET]
' 3軸BP補間データ
Dim Data3Bp (1) As DATA_3BP
Data3Bp(0).Bp1p = &HFF30S
Data3Bp(0).Bp1m = 0
Data3Bp(0).Bp2p = 0
Data3Bp(0).Bp2m = &H84FFS
Data3Bp(0).Bp3p = 0
Data3Bp(0).Bp3m = &HAC35S

Data3Bp(1).Bp1p = &H0xAC35S
Data3Bp(1).Bp1m = 0
Data3Bp(1).Bp2p = &HC000S
Data3Bp(1).Bp2m = &H36E7S
Data3Bp(1).Bp3p = &HC000S
Data3Bp(1).Bp3m = &H3F3FS

Call Nmc_IPMode(No, IcNo, &H7) ' 補間モード設定。X, Y, Z軸指定。

' 速度関連パラメータ設定(実際の設定記述は省略)

' 3軸BP補間実行。データ数2, X, Y, Z軸
Ret = Nmc_3BPExecMC8500P_BG(Handle.ToInt32, No, IcNo, Data3Bp, 2, &H7)
If Ret = BP_START Then ' 戻り値正常(補間開始)
Call MsgBox("補間開始")
End If
End Sub

' WM_BP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)
Select Case (m.Msg)
Case WM_BP_END ' BP補間終了メッセージ
If m.LParam.ToInt32 = BP_END Then ' 戻り値正常(補間終了)
Call MsgBox("補間正常終了")
End If
Exit Select
End Select
MyBase.WndProc(m)
End Sub

[C#]
DATA_3BP [] Data3Bp = new DATA_3BP[2]; // 3軸BP補間データ
// 補間データ設定
Data3Bp[0].Bp1p = 0xFF30; // 1111 1111 0011 0000 BP1+方向 10パルス
Data3Bp[0].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data3Bp[0].Bp2p = 0; // 0000 0000 0000 0000 BP2+方向 0パルス
Data3Bp[0].Bp2m = 0x84FF; // 1000 0100 1111 1111 BP2-方向 10パルス
Data3Bp[0].Bp3p = 0; // 0000 0000 0000 0000 BP3+方向 0パルス
Data3Bp[0].Bp3m = 0xAC35; // 1010 1100 0011 0101 BP3-方向 8パルス

Data3Bp[1].Bp1p = 0xAC35; // 1010 1100 0011 0101 BP1+方向 8パルス
Data3Bp[1].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data3Bp[1].Bp2p = 0xC000; // 1100 0000 0000 0000 BP2+方向 2パルス
Data3Bp[1].Bp2m = 0x36E7; // 0011 0110 1110 0111 BP2-方向 10パルス
Data3Bp[1].Bp3p = 0xC000; // 1100 0000 0000 0000 BP3+方向 2パルス
Data3Bp[1].Bp3m = 0x3F3F; // 0011 1111 0011 1111 BP3-方向 12パルス

```

```

MC8000P.Nmc_IPMode(No, IcNo, 0x0007); //補間モード設定。X, Y, Z軸指定。
// 速度関連パラメータ設定(実際の設定記述は省略)
// 3軸BP補間バックグラウンド実行
Ret = MC8000P.Nmc_3BPExecMC8500P_BG((System.IntPtr)parent.Handle, No, IcNo, Data3Bp, 2, 0x7);

if(Ret == Nmc_Status.BP_START) // 補間開始正常の場合

//WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
// 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
base.WndProc ( ref m );
if ( m.Msg == (int)MSG_ID.WM_BP_END )
{
if((uint)m.LParam == Nmc_Status.BP_END) // BP補間処理正常終了
}
}

if(Ret == Nmc_Status.BP_END) // 戻り値正常

```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定して下さい。


```

// 速度関連パラメータ設定(実際の設定記述は省略)

Ret = Nmc_4BPExecMC8500P_BG(hWnd, No, IcNo, Data4Bp, 2, 0xF);
//4軸BP補間実行。データ数2, X, Y, Z, U軸
if(Ret == BP_START) AfxMessageBox("補間開始"); //戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog) // WM_BP_ENDメッセージ受信関数設定
ON_MESSAGE(WM_BP_END, OnMsg_BP)
END_MESSAGE_MAP()

// WM_BP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_BP(WPARAM BoardNo, LPARAM Status)
{
if(Status == BP_END) AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
return 0;
}

[VB.NET]
' 4軸BP補間データ
Dim Data4Bp(1) As DATA_4BP
Data4Bp(0).Bp1p = &HFFE4S
Data4Bp(0).Bp1m = 0
Data4Bp(0).Bp2p = &H3FFS
Data4Bp(0).Bp2m = &H4000S
Data4Bp(0).Bp3p = 0
Data4Bp(0).Bp3m = &HFFFFS
Data4Bp(0).Bp4p = &HFE80S
Data4Bp(0).Bp4m = &H000FS

Data4Bp(1).Bp1p = 0
Data4Bp(1).Bp1m = &H3FFS
Data4Bp(1).Bp2p = &HFFD0S
Data4Bp(1).Bp2m = 0
Data4Bp(1).Bp3p = &H4AABS
Data4Bp(1).Bp3m = 0
Data4Bp(1).Bp4p = &H1FFFS
Data4Bp(1).Bp4m = 0
Call Nmc_IPMode(No, IcNo, &HF) ' 補間モード設定。X, Y, Z, U軸指定。

' 速度関連パラメータ設定(実際の設定記述は省略)

' 4軸BP補間実行。データ数2, X, Y, Z, U軸
Ret = Nmc_4BPExecMC8500P_BG(Handle.ToInt32, No, IcNo, Data4Bp, 2, &HF)
If Ret = BP_START Then ' 戻り値正常(補間開始)
Call MsgBox("補間開始")
End If
End Sub

' WM_BP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)
Select Case (m.Msg)
Case WM_BP_END ' BP補間終了メッセージ
If m.LParam.ToInt32 = BP_END Then ' 戻り値正常(補間終了)
Call MsgBox("補間正常終了")
End If
Exit Select
End Select
MyBase.WndProc(m)
End Sub

[C#]
DATA_4BP [] Data4Bp = new DATA_4BP[2]; // 4軸BP補間データ
// 補間データ設定
Data4Bp[0].Bp1p = 0xFFE4; // 1111 1111 1110 0100 BP1+方向 12パルス
Data4Bp[0].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data4Bp[0].Bp2p = 0x03FF; // 0000 0011 1111 1111 BP2+方向 10パルス
Data4Bp[0].Bp2m = 0x4000; // 0100 0000 0000 0000 BP2-方向 1パルス
Data4Bp[0].Bp3p = 0; // 0000 0000 0000 0000 BP3+方向 0パルス
Data4Bp[0].Bp3m = 0xFFFF; // 1111 1111 1111 1111 BP3-方向 16パルス
Data4Bp[0].Bp4p = 0xFE80; // 1111 1110 1000 0000 BP4+方向 8パルス
Data4Bp[0].Bp4m = 0x000F; // 0000 0000 0000 1111 BP4-方向 4パルス

```

```

Data4Bp[1].Bp1p = 0;           // 0000 0000 0000 0000    BP1+方向    0パルス
Data4Bp[1].Bp1m = 0x03FF;     // 0000 0011 1111 1111    BP1-方向    10パルス
Data4Bp[1].Bp2p = 0xFFD0;     // 1111 1111 1101 0000    BP2+方向    11パルス
Data4Bp[1].Bp2m = 0;         // 0000 0000 0000 0000    BP2-方向    0パルス
Data4Bp[1].Bp3p = 0x4AAB;     // 0100 1010 1010 1011    BP3+方向    8パルス
Data4Bp[1].Bp3m = 0;         // 0000 0000 0000 0000    BP3-方向    0パルス
Data4Bp[1].Bp4p = 0x1FFF;     // 0001 1111 1111 1111    BP4+方向    13パルス
Data4Bp[1].Bp4m = 0;         // 0000 0000 0000 0000    BP4-方向    0パルス

MC8000P.Nmc_IPMode(No, IcNo, 0x000F);           //補間モード設定。X, Y, Z, U軸指定。

// 速度関連パラメータ設定(実際の設定記述は省略)

// 4軸BP補間バックグラウンド実行
Ret = MC8000P.Nmc_4BPExecMC8500P_BG((System.IntPtr)parent.Handle, No, IcNo, Data3Bp, 2, 0xF);

if(Ret == Nmc_Status.BP_START)                 // 補間開始正常の場合

//WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    // 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
    base.WndProc ( ref m );
    if ( m.Msg == (int)MSG_ID.WM_BP_END )
    {
        if((uint)m.LParam == Nmc_Status.BP_END) // BP補間処理正常終了
        }
    }

    if(Ret == Nmc_Status.BP_END)                 // 戻り値正常

```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定して下さい。


```
// 速度関連パラメータ設定
Nmc_StartSpd(No, IcNo, AXIS_X, 4000000); // 定速ドライブにするため4M
// 他の設定記述は省略
Ret = Nmc_2CIPExecMC8500P(No, IcNo, Data2Cip, 2, 0x3); // 2軸連続補間実行。データ数2, X, Y軸

if(Ret == CIP_END) AfxMessageBox("正常終了"); // 戻り値正常
```

[VB.NET]

```
' 2軸補間データ
Dim Data2Cip(1) As DATA_2CIP_MC8500P
Data2Cip(0).Command = MC8500P_CMD_2CIP ' 2軸直線補間
Data2Cip(0).Speed = 0
Data2Cip(0).EndP1 = 4500
Data2Cip(0).EndP2 = 0
Data2Cip(0).Center1 = 0
Data2Cip(0).Center2 = 0

Data2Cip(1).Command = MC8500P_CMD_CIPCCW ' CCW 円弧補間
Data2Cip(1).Speed = 0
Data2Cip(1).EndP1 = 1500
Data2Cip(1).EndP2 = 1500
Data2Cip(1).Center1 = 0
Data2Cip(1).Center2 = 1500

Call Nmc_IPMode(No, IcNo, &H3) ' 補間モード設定。X, Y軸指定。

' 速度関連パラメータ設定
Call Nmc_StartSpd(No, IcNo, AXIS_X, 4000000) ' 定速ドライブにするため4M
' 他の設定記述は省略

' 2軸連続補間実行。データ数2, X, Y軸
Ret = Nmc_2CIPExecMC8500P(No, IcNo, Data2Cip, 2, &H3, False)

If Ret = CIP_END Then ' 戻り値正常
    Call MsgBox("正常終了")
End If
```

[C#]

```
DATA_2CIP_MC8500P [] Data2Cip = new DATA_2CIP_MC8500P[2]; // 2軸連続補間データ
// 補間データ設定
Data2Cip[0].Command = (ushort)CMD.MC8500P_CMD_2CIP; // 2軸直線補間
Data2Cip[0].EndP1 = 0;
Data2Cip[0].EndP2 = 4500;
Data2Cip[0].Center1 = 0;
Data2Cip[0].Center2 = 0;
Data2Cip[0].Speed = 0;

Data2Cip[1].Command = (ushort)CMD.MC8500P_CMD_CIPCCW; // CCW円弧補間
Data2Cip[1].EndP1 = 1500;
Data2Cip[1].EndP2 = 1500;
Data2Cip[1].Center1 = 0;
Data2Cip[1].Center2 = 1500;
Data2Cip[1].Speed = 0;

MC8000P.Nmc_IPMode(No, IcNo, 0x0003) // 補間モード設定。X, Y軸指定。

// 2軸連続補間実行
// この関数は補間が終了するまで制御が戻りません
Ret = MC8000P.Nmc_2CIPExecMC8500P(No, IcNo, Data2Cip, 2, 0x3, SpdChgFlg);
if(Ret == Nmc_Status.CIP_END) // 連続補間処理正常終了
```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定してください。


```

Data3Cip[1].EndP1 = 2000;
Data3Cip[1].EndP2 = -1000;
Data3Cip[1].EndP3 = 3000;
Data3Cip[1].Speed = 0;

Nmc_IPMode(No, IcNo, 0x0007); //補間モード設定。X, Y, Z軸指定。

// 速度関連パラメータ設定
Nmc_StartSpd(No, IcNo, AXIS_X, 4000000); // 定速ドライブにするため4M
// 他の設定記述は省略

Ret = Nmc_3CIPExecMC8500P(No, IcNo, Data3Cip, 2, 0x7);
// 3軸連続補間実行。データ数2, X, Y, Z軸
if(Ret == CIP_END) AfxMessageBox("正常終了"); //戻り値正常

```

[VB. NET]

```

' 3軸連続補間データ設定
Dim Data3Cip(1) As DATA_3CIP_MC8500P
Data3Cip(0).EndP1 = 1000
Data3Cip(0).EndP2 = 2000
Data3Cip(0).EndP3 = 3000
Data3Cip(0).Speed = 0

Data3Cip(1).EndP1 = 2000
Data3Cip(1).EndP2 = -1000
Data3Cip(1).EndP3 = 3000
Data3Cip(1).Speed = 0

Call Nmc_IPMode(No, IcNo, &H7) '補間モード設定。X, Y, Z軸指定。

' 速度関連パラメータ設定
Call Nmc_StartSpd(No, IcNo, AXIS_X, 4000000) '定速ドライブにするため4M
' 他の設定記述は省略

' 3軸連続補間実行。データ数2, X, Y, Z軸

Ret = Nmc_3CIPExecMC8500P(No, IcNo, Data3Cip, 2, &H7, False)

If Ret = CIP_END Then '戻り値正常
    Call MsgBox("正常終了")
End If

```

[C#]

```

DATA_3CIP_MC8500P [] Data3Cip = new DATA_3CIP_MC8500P[2]; // 3軸連続補間データ用
// 補間データ設定
Data3Cip[0].EndP1 = 1000;
Data3Cip[0].EndP2 = 2000;
Data3Cip[0].EndP3 = 3000;
Data3Cip[0].Speed = 0;

Data3Cip[1].EndP1 = 2000;
Data3Cip[1].EndP2 = -1000;
Data3Cip[1].EndP3 = 3000;
Data3Cip[1].Speed = 0;

MC8000P.Nmc_IPMode(No, IcNo, 0x0007) //補間モード設定。X, Y, Z軸指定。

// 3軸連続補間実行
// この関数は補間が終了するまで制御が戻りません
Ret = MC8000P.Nmc_3CIPExecMC8500P(No, IcNo, Data3Cip, 2, 0x7, SpdChgFlg);
if(Ret == Nmc_Status.CIP_END) // 連続補間処理正常終了

```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定して下さい。

関数名	機能 及び 内容
Nmc_4CIPExecMC8500P	<p>指定した補間データで4軸連続補間を実行する。この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。</p> <p>VC DWORD Nmc_4CIPExecMC8500P(int No, int IcNo, DATA_4CIP_MC8500P* pData4Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE);</p> <p>VB.NET Function Nmc_4CIPExecMC8500P(ByVal No As Integer, ByVal IcNo As Integer, ByVal pData4Cip As DATA_4CIP_MC8500P(), ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer) As Integer</p> <p>C# Nmc_Status MC8000P.Nmc_4CIPExecMC8500P(int No, IcNo, DATA_4CIP_MC8500P[] pData4Cip, int DataCnt, int IpAxis, bool SpdChgFlg);</p> <p>入力パラメータ</p> <p>No ボード番号 (ボード上のロータリスイッチの値 (0~15))</p> <p>IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照</p> <p>pData4Cip 4軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス。 (DATA_4CIP_MC8500Pのアドレス)。 DATA_4CIP_MC8500Pに実行する補間データをセットし、アドレスを指定する。 DATA_4CIP_MC8500Pについては「5.1.3 補足説明」(3)参照。</p> <p>DataCnt 4軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。</p> <p>IpAxis 補間を実行する軸。補間モード設定(2A)HのD0~D3(軸指定)の設定値と同じ値を指定する。 「5.1.3 補足説明」(4)参照。</p> <p>SpdChgFlg 補間実行中に速度を変更するかどうかを設定する。 変更する場合は「5.1.3 補足説明」(5)参照。 [VC] TRUE : 変更する、FALSE : 変更しない。省略可能。省略時FALSE。 [VB.NET] True : 変更する、False : 変更しない [C#] true : 続ける、false : 続けない 変更する選択時 : DATA_4CIP_MC8500PのSpeedに設定した値を参照します。 Speedに1~4000000の値設定・・・設定した速度に変更します。 Speedに0 設定・・・速度は変更しません。 変更しない選択時 : DATA_4CIP_MC8500PのSpeedに設定した値を参照しません。</p> <p>戻り値</p> <p>補間処理が正常終了した場合は CIP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は「5.1.3 補足説明」(1)参照。</p> <p>■ 正常終了 CIP_END 連続補間処理正常終了</p> <p>■ エラーコード</p> <p>CIP_CNT_ERR 指定されたデータ数が範囲外</p> <p>CIP_ALREADY_EXEC 既にB P補間、あるいは連続補間が実行中</p> <p>CIP_MALLOC_ERR メモリを確保できなかった</p> <p>CIP_CMD_ERR コマンドエラー (ユーザーが指定したコマンドが間違っている)</p> <p>CIP_PARAM_ERR 引数の値が正しくない</p> <p>CIP_NOT_OPEN_ERR 指定したボードがオープンされていない</p> <p>CIP_OTHER_ERR その他のエラー</p> <p>CIP_FUNC_ERR 使用できない関数を使用</p> <p>CIP_STOP 連続補間が途中で停止した</p> <p>CIP_USER_STOP 連続補間実行中にユーザーが中断した</p> <p>CIP_DRIVE_ERR 連続補間実行中にボードでエラー発生 (RR0にエラー情報がセットされた)</p> <p>CIP_STOP_CERR 連続補間が途中で停止した (RR2エラーによる停止の場合)</p> <p>CIP_GETSC_ERR スタックカウンタ読み出しエラー</p> <p>注意</p> <p>この関数を実行する前に、初速度を4,000,000に設定して下さい。(本関数実行中に初速度を変更しないで下さい。)詳細は「5.1.4 使用方法」参照。</p> <p>使用例</p> <p>[VC]</p> <pre>DATA_4CIP_MC8500P Data4Cip[2]; // 4軸連続補間データ用 // 4軸連続補間データ設定 Data4Cip[0].EndP1 = 1000; Data4Cip[0].EndP2 = 2000; Data4Cip[0].EndP3 = 3000;</pre>

```

Data4Cip[0].EndP4 = 4000;
Data4Cip[0].Speed = 0;

Data4Cip[1].EndP1 = 2000;
Data4Cip[1].EndP2 = -1000;
Data4Cip[1].EndP3 = 3000;
Data4Cip[1].EndP4 = -2000;
Data4Cip[1].Speed = 0;

Nmc_IPMode(No, IcNo, 0x000F); //補間モード設定。X, Y, Z, U軸指定。

// 速度関連パラメータ設定
Nmc_StartSpd(No, IcNo, AXIS_X, 4000000); // 定速ドライブにするため4M
// 他の設定記述は省略
Ret = Nmc_4CIPExecMC8500P(No, IcNo, Data4Cip, 2, 0xF);
// 4軸連続補間実行。データ数2, X, Y, Z, U軸
if(Ret == CIP_END) AfxMessageBox("正常終了"); //戻り値正常

```

[VB.NET]

```

' 4軸連続補間データ設定
Dim Data4Cip(1) As DATA_4CIP_MC8500P
Data4Cip(0).EndP1 = 1000
Data4Cip(0).EndP2 = 2000
Data4Cip(0).EndP3 = 3000
Data4Cip(0).EndP4 = 4000
Data4Cip(0).Speed = 0

Data4Cip(1).EndP1 = 2000
Data4Cip(1).EndP2 = -1000
Data4Cip(1).EndP3 = 3000
Data4Cip(1).EndP4 = -2000
Data4Cip(1).Speed = 0

Call Nmc_IPMode(No, IcNo, &HF) '補間モード設定。X, Y, Z, U軸指定。

' 速度関連パラメータ設定
Call Nmc_StartSpd(No, IcNo, AXIS_X, 4000000) '定速ドライブにするため4M
' 他の設定記述は省略

' 4軸連続補間実行。データ数2, X, Y, Z, U軸
Ret = Nmc_4CIPExecMC8500P(No, IcNo, Data4Cip, 2, &HF, False)

If Ret = CIP_END Then '戻り値正常
    Call MsgBox("正常終了")
End If

```

[C#]

```

DATA_4CIP_MC8500P [] Data4Cip = new DATA_4CIP_MC8500P[2]; // 4軸連続補間データ用
// 補間データ設定
Data4Cip[0].EndP1 = 1000;
Data4Cip[0].EndP2 = 2000;
Data4Cip[0].EndP3 = 3000;
Data4Cip[0].EndP4 = 3000;
Data4Cip[0].Speed = 0;

Data4Cip[1].EndP1 = 2000;
Data4Cip[1].EndP2 = -1000;
Data4Cip[1].EndP3 = 3000;
Data4Cip[1].EndP4 = -2000;
Data4Cip[1].Speed = 0;

MC8000P.Nmc_IPMode(No, IcNo, 0x000F) //補間モード設定。X, Y, Z, U軸指定。

// 4軸連続補間実行
// この関数は補間が終了するまで制御が戻りません
Ret = MC8000P.Nmc_4CIPExecMC8500P(No, IcNo, Data4Cip, 2, 0xF, SpdChgFlg);
if(Ret == Nmc_Status.CIP_END) // 連続補間処理正常終了

```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定してください。

関数名	機能 及び 内容
Nmc_2CIPExecMC8500P_BG	<p>指定した補間データで2軸連続補間をバックグラウンドで実行する。 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_CIP_ENDメッセージを送信し、終了ステータスを渡します。</p> <p>VC DWORD Nmc_2CIPExecMC8500P_BG(HWND User_hWnd, int No, int IcNo, DATA_2CIP_MC8500P* pData2Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE);</p> <p>VB.NET Function Nmc_2CIPExecMC8500P_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByVal pData2Cip As DATA_2CIP_MC8500P(), ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer) As Integer</p> <p>C# Nmc_Status MC8000P.Nmc_2CIPExecMC8500P_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_2CIP_MC8500P[] pData2Cip, int DataCnt, int IpAxis, bool SpdChgFlg);</p> <p>入力パラメータ</p> <p>User_hWnd ユーザーアプリケーションのウィンドウハンドル No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 pData2Cip 2軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス (DATA_2CIP_MC8500Pのアドレス)。 DATA_2CIP_MC8500Pに実行する補間データをセットし、アドレスを指定する。 DATA_2CIP_MC8500Pについては「5.1.3 補足説明」(3)参照。 DataCnt 2軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis 補間を実行する軸。補間モード設定(2A)HのD0~D3(軸指定)の設定値と同じ値を指定する。 「5.1.3 補足説明」(4)参照。 SpdChgFlg 補間実行中に速度を変更するかどうかを設定する。 変更する場合は「5.1.3 補足説明」(5)参照。 [VC] TRUE : 変更する、FALSE : 変更しない。省略可能。省略時FALSE。 [VB.NET] True : 変更する、False : 変更しない [C#] true : 変更する、false : 変更しない 変更する選択時 : DATA_2CIP_MC8500PのSpeedに設定した値を参照します。 Speedに1~4000000の値設定・・・設定した速度に変更します。 Speedに0 設定・・・速度は変更しません。 変更しない選択時 : DATA_2CIP_MC8500PのSpeedに設定した値を参照しません。</p> <p>戻り値</p> <p>バックグラウンドで補間処理が正常に開始した場合は CIP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は「5.1.3 補足説明」(1)参照。</p> <p>■ 正常開始 CIP_START バックグラウンドで連続補間処理が正常に開始した</p> <p>■ エラーコード(補間開始前のエラー)</p> <p>CIP_CNT_ERR 指定されたデータ数が範囲外 CIP_ALREADY_EXEC 既にBP補間、あるいは連続補間が実行中 CIP_THREAD_ERR スレッドを起動できなかった CIP_MALLOC_ERR メモリを確保できなかった CIP_CMD_ERR コマンドエラー (ユーザーが指定したコマンドが間違っている) CIP_PARAM_ERR 引数の値が正しくない CIP_NOT_OPEN_ERR 指定したボードがオープンされていない CIP_OTHER_ERR その他のエラー CIP_FUNC_ERR 使用できない関数を使用</p> <p>バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、WM_CIP_ENDメッセージを送信します。WM_CIP_ENDのメッセージ受信関数で受け取る第1引数にボード番号を、第2引数に終了ステータスを渡します。その終了ステータスは、補間が正常終了した場合はCIP_ENDです。補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。</p> <p>■ 正常終了 CIP_END 連続補間処理正常終了</p> <p>■ エラーコード(補間開始後のエラー)</p> <p>CIP_USER_STOP 連続補間実行中にユーザーが中断した CIP_DRIVE_ERR 連続補間実行中にボードでエラー発生 (RROにエラー情報がセットされた) CIP_STOP_CERR 連続補間が途中で停止した (RR2エラーによる停止の場合) CIP_GETSC_ERR スタックカウンタ読み出しエラー</p> <p>注意</p>

この関数を実行する前に、初速度を 4,000,000 に設定して下さい。（本関数実行中に初速度を変更しないで下さい。）詳細は「5.1.4 使用方法」参照。

使用例

```
[VC]
{
    // 2軸連続補間データ          コマンド          速度  終点1  終点2  中心点1  中心点2
    DATA_2CIP_MC8500P Data2Cip[2]={MC8500P_CMD_2CIP, 0, 4500, 0, 0, 0}, //2軸直線補間
                                   {MC8500P_CMD_CIPCCW, 0, 1500, 1500, 0, 1500}}; //CCW円弧補間

    Nmc_IPMode(No, IcNo, 0x0003); //補間モード設定。X, Y軸指定。

    // 速度関連パラメータ設定
    Nmc_StartSpd(No, IcNo, AXIS_X, 4000000); // 定速ドライブにするため4M
                                              // 他の設定記述は省略

    Ret = Nmc_2CIPExecMC8500P_BG(hWnd, No, IcNo, Data2Cip, 2, 0x3);
                                              //2軸連続補間実行。データ数2, X, Y軸

    if(Ret == CIP_START) AfxMessageBox("補間開始"); //戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog) // WM_CIP_ENDメッセージ受信関数設定
    ON_MESSAGE(WM_CIP_END, OnMsg_CIP)
END_MESSAGE_MAP()

// WM_CIP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_CIP(WPARAM BoardNo, LPARAM Status)
{
    if(Status == CIP_END) AfxMessageBox("補間正常終了"); // 戻り値正常(補間終了)
    return 0;
}

[VB.NET]
' 2軸連続補間データ
Dim Data2Cip(1) As DATA_2CIP_MC8500P
Data2Cip(0).Command = MC8500P_CMD_2CIP ' 2軸直線補間
Data2Cip(0).Speed = 0 ' 速度変更する
Data2Cip(0).EndP1 = 4500 ' 終点1
Data2Cip(0).EndP2 = 0 ' 終点2
Data2Cip(0).Center1 = 0 ' 中心点1
Data2Cip(0).Center2 = 0 ' 中心点2

Data2Cip(1).Command = MC8500P_CMD_CIPCCW ' CCW円弧補間
Data2Cip(1).Speed = 0 ' 速度変更する
Data2Cip(1).EndP1 = 1500 ' 終点1
Data2Cip(1).EndP2 = 1500 ' 終点2
Data2Cip(1).Center1 = 0 ' 中心点1
Data2Cip(1).Center2 = 1500 ' 中心点2

Call Nmc_IPMode(No, IcNo, &H3) ' 補間モード設定。X, Y軸指定。

' 速度関連パラメータ設定
Call Nmc_StartSpd(No, IcNo, AXIS_X, 4000000) ' 定速ドライブにするため4M
                                              ' 他の設定記述は省略

' 2軸連続補間実行。データ数2, X, Y軸
Ret = Nmc_2CIPExecMC8500P_BG(Handle.ToInt32, No, IcNo, Data2Cip, 2, &H3, False)

If Ret = CIP_START Then ' 戻り値正常(補間開始)
    Call MsgBox("補間開始")
End If
End Sub

' WM_CIP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)
    Select Case (m.Msg)
        Case WM_CIP_END ' 連続補間終了メッセージ
            If m.LParam.ToInt32 = CIP_END Then ' 戻り値正常(補間終了)
                Call MsgBox("補間正常終了")
            End If
            Exit Select
    End Select
    MyBase.WndProc(m)
End Sub

[C#]
```

```

DATA_2CIP_MC8500P [] Data2Cip = new DATA_2CIP_MC8500P[2]; // 2軸連続補間データ
// 補間データ設定
Data2Cip[0].Command = (ushort)CMD.MC8500P_CMD_2CIP; // 2軸直線補間
Data2Cip[0].EndP1 = 0;
Data2Cip[0].EndP2 = 4500;
Data2Cip[0].Center1 = 0;
Data2Cip[0].Center2 = 0;
Data2Cip[0].Speed = 0;

Data2Cip[1].Command = (ushort)CMD.MC8500P_CMD_CIPCCW; // CCW円弧補間
Data2Cip[1].EndP1 = 1500;
Data2Cip[1].EndP2 = 1500;
Data2Cip[1].Center1 = 0;
Data2Cip[1].Center2 = 1500;
Data2Cip[1].Speed = 0;

MC8000P.Nmc_IPMode(No, IcNo, 0x3); //補間モード設定。X, Y軸指定。

// 2軸連続補間実行
// この関数は補間が終了するまで制御が戻りません
Ret = MC8000P.Nmc_2CIPExecMC8500P_BG(No, IcNo, Data2Cip, 2, 0x3, SpdChgFlg);

// WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    // 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
    base.WndProc ( ref m );
    if ( m.Msg == (int)MSG_ID.WM_CIP_END )
    {
        if((uint)m.LParam == Nmc_Status.CIP_END) // 連続補間処理正常終了
        }
    }
}

```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定して下さい。

注意

この関数を実行する前に、初速度を 4,000,000 に設定して下さい。（本関数実行中に初速度を変更しないで下さい。）詳細は「5.1.4 使用方法」参照。

使用例

[VC]

```
{
    DATA_3CIP_MC8500P Data3Cip[2];           // 3軸連続補間データ用
    // 3軸連続補間データ設定
    Data3Cip[0].EndP1 = 1000;
    Data3Cip[0].EndP2 = 2000;
    Data3Cip[0].EndP3 = 3000;

    Data3Cip[1].EndP1 = 2000;
    Data3Cip[1].EndP2 = -1000;
    Data3Cip[1].EndP3 = 3000;

    Nmc_IPMode(No, IcNo, 0x0007);           //補間モード設定。X, Y, Z軸指定。

    // 速度関連パラメータ設定
    Nmc_StartSpd(No, IcNo, AXIS_X, 4000000); // 定速ドライブにするため4M
                                              // 他の設定記述は省略
    Ret = Nmc_3CIPExecMC8500P_BG(hWnd, No, IcNo, Data3Cip, 2, 0x7);
                                              //3軸連続補間実行。データ数2, X, Y, Z軸
    if(Ret == CIP_START) AfxMessageBox("補間開始");//戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog) // WM_CIP_ENDメッセージ受信関数設定
    ON_MESSAGE(WM_CIP_END, OnMsg_CIP)
END_MESSAGE_MAP()

// WM_CIP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_CIP(WPARAM BoardNo, LPARAM Status)
{
    if(Status == CIP_END) AfxMessageBox("補間正常終了");// 戻り値正常(補間終了)
    return 0;
}
```

[VB.NET]

```
' 3軸連続補間データ設定
Dim Data3Cip(1) As DATA_3CIP_MC8500P
Data3Cip(0).EndP1 = 10000
Data3Cip(0).EndP2 = 20000
Data3Cip(0).EndP3 = 30000
Data3Cip(0).Speed = 0

Data3Cip(1).EndP1 = 2000
Data3Cip(1).EndP2 = -1000
Data3Cip(1).EndP3 = 3000
Data3Cip(1).Speed = 0

Call Nmc_IPMode(No, IcNo, &H7)           ' 補間モード設定。X, Y, Z軸指定。

' 速度関連パラメータ設定
Call Nmc_StartSpd(No, IcNo, AXIS_X, 4000000) ' 定速ドライブにするため4M
                                              ' 他の設定記述は省略

' 3軸連続補間実行。データ数2, X, Y, Z軸
Ret = Nmc_3CIPExecMC8500P_BG(Handle.ToInt32, No, IcNo, Data3Cip, 2, &H7, False)
If Ret = CIP_START Then                    ' 戻り値正常(補間開始)
    Call MsgBox("補間開始")
End If
End Sub

' WM_CIP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)
    Select Case (m.Msg)
        Case WM_CIP_END                    ' 連続補間終了メッセージ
            If m.LParam.ToInt32 = CIP_END Then ' 戻り値正常(補間終了)
                Call MsgBox("補間正常終了")
            End If
        Exit Select
    End Select
    MyBase.WndProc(m)
End Sub
```

```

[C#]
DATA_3CIP_MC8500P [] Data3Cip = new DATA_3CIP_MC8500P[2];    // 3軸連続補間データ用
// 補間データ設定
Data3Cip[0].EndP1 = 1000;
Data3Cip[0].EndP2 = 2000;
Data3Cip[0].EndP3 = 3000;
Data3Cip[0].Speed = 0;

Data3Cip[1].EndP1 = 2000;
Data3Cip[1].EndP2 = -1000;
Data3Cip[1].EndP3 = 3000;
Data3Cip[1].Speed = 0;

MC8000P.Nmc_IPMode(No, IcNo, 0x0007);           //補間モード設定。X, Y, Z軸指定。

// 3軸連続補間実行
// バックグラウンドで実行する
Ret = MC8000P.Nmc_3CIPExec_BG((System.IntPtr)parent.Handle, gBoardNo, int IcNo,
                               Data3Cip, 2, 0x7, SpdChgFlg, ContinueFlg);
if(Ret == Nmc_Status.CIP_START)                // 連続補間処理正常開始

//WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    // 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
    base.WndProc ( ref m );
    if ( m.Msg == (int)MSG_ID.WM_CIP_END )
    {
        if((uint)m.LParam == Nmc_Status.CIP_END)    // 連続補間処理正常終了
        }
    }
}

```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定して下さい。

関数名	機能 及び 内容
Nmc_4CIPExecMC8500P_BG	<p>指定した補間データで4軸連続補間をバックグラウンドで実行する。 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_CIP_ENDメッセージを送信し、終了ステータスを渡します。</p> <p>VC DWORD Nmc_4CIPExecMC8500P_BG(HWND User_hWnd, int No, int IcNo, DATA_4CIP_MC8500P* pData4Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE);</p> <p>VB.NET Function Nmc_4CIPExecMC8500P_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByVal pData4Cip As DATA_4CIP_MC8500P(), ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer) As Integer</p> <p>C# Nmc_Status MC8000P.Nmc_4CIPExecMC8500P_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_4CIP_MC8500P[] pData4Cip, int IpAxis, bool SpdChgFlg);</p> <p>入力パラメータ</p> <p>User_hWnd ユーザーアプリケーションのウィンドウハンドル No ボード番号 (ボード上のロータリースイッチの値 (0~15)) IcNo IC番号 (0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照 pData4Cip 4軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス (DATA_4CIP_MC8500Pのアドレス)。 DATA_4CIP_MC8500Pに実行する補間データをセットし、アドレスを指定する。 DATA_4CIP_MC8500Pについては「5.1.3 補足説明」(3)参照。 DataCnt 4軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis 補間を実行する軸。補間モード設定(2A)HのD0~D3(軸指定)の設定値と同じ値を指定する。 「5.1.3 補足説明」(4)参照。 SpdChgFlg 補間実行中に速度を変更するかどうかを設定する。 変更する場合は「5.1.3 補足説明」(5)参照。 [VC] TRUE : 変更する、FALSE : 変更しない。省略可能。省略時FALSE。 [VB.NET] True : 変更する、False : 変更しない [C#] true : 変更する、false : 変更しない 変更する選択時 : DATA_4CIP_MC8500PのSpeedに設定した値を参照します。 Speedに1~4000000の値設定・・・設定した速度に変更します。 Speedに0 設定・・・速度は変更しません。 変更しない選択時 : DATA_4CIP_MC8500PのSpeedに設定した値を参照しません。</p> <p>戻り値</p> <p>バックグラウンドで補間処理が正常に開始した場合は CIP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は「5.1.3 補足説明」(1)参照。</p> <p>■ 正常開始 CIP_START バックグラウンドで連続補間処理が正常に開始した</p> <p>■ エラーコード(補間開始前のエラー)</p> <p>CIP_CNT_ERR 指定されたデータ数が範囲外 CIP_ALREADY_EXEC 既にBP補間、あるいは連続補間が実行中 CIP_THREAD_ERR スレッドを起動できなかった CIP_MALLOC_ERR メモリを確保できなかった CIP_CMD_ERR コマンドエラー (ユーザーが指定したコマンドが間違っている) CIP_PARAM_ERR 引数の値が正しくない CIP_NOT_OPEN_ERR 指定したボードがオープンされていない CIP_OTHER_ERR その他のエラー CIP_FUNC_ERR 使用できない関数を使用</p> <p>バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、WM_CIP_ENDメッセージを送信します。WM_CIP_ENDのメッセージ受信関数で受け取る第1引数にボード番号を、第2引数に終了ステータスを渡します。その終了ステータスは、補間が正常終了した場合はCIP_ENDです。補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。</p> <p>■ 正常終了 CIP_END 連続補間処理正常終了</p> <p>■ エラーコード(補間開始後のエラー)</p> <p>CIP_USER_STOP 連続補間実行中にユーザーが中断した CIP_DRIVE_ERR 連続補間実行中にボードでエラー発生 (RR0にエラー情報がセットされた) CIP_STOP_CERR 連続補間が途中で停止した (RR2エラーによる停止の場合) CIP_GETSC_ERR スタックカウンタ読み出しエラー</p>

注意

この関数を実行する前に、初速度を 4,000,000 に設定して下さい。（本関数実行中に初速度を変更しないで下さい。）詳細は「5.1.4 使用方法」参照。

使用例

[VC]

```
{
    DATA_4CIP_MC8500P Data4Cip[2];           // 4軸連続補間データ用
    // 4軸連続補間データ設定
    Data4Cip[0].EndP1 = 1000;
    Data4Cip[0].EndP2 = 2000;
    Data4Cip[0].EndP3 = 3000;
    Data4Cip[0].EndP4 = 4000;

    Data4Cip[1].EndP1 = 2000;
    Data4Cip[1].EndP2 = -1000;
    Data4Cip[1].EndP3 = 3000;
    Data4Cip[1].EndP4 = 4000;

    Nmc_IPMode(No, IcNo, 0x000F);           //補間モード設定。X, Y, Z, U軸指定。

    // 速度関連パラメータ設定
    Nmc_StartSpd(No, IcNo, AXIS_X, 4000000); // 定速ドライブにするため4M
                                              // 他の設定記述は省略
    Ret = Nmc_4CIPExecMC8500P_BG(hWnd, No, IcNo, Data4Cip, 2, 0xF);
                                              //4軸連続補間実行。データ数2, X, Y, Z, U軸
    if(Ret == CIP_START) AfxMessageBox("補間開始");//戻り値正常(補間開始)
}

BEGIN_MESSAGE_MAP(CMC_SAMPLEDlg, CDialog) // WM_CIP_ENDメッセージ受信関数設定
    ON_MESSAGE(WM_CIP_END, OnMsg_CIP)
END_MESSAGE_MAP()

// WM_CIP_ENDメッセージ受信関数
afx_msg LRESULT CMC_SAMPLEDlg::OnMsg_CIP(WPARAM BoardNo, LPARAM Status)
{
    if(Status == CIP_END) AfxMessageBox("補間正常終了");// 戻り値正常(補間終了)
    return 0;
}
}
```

[VB.NET]

```
' 4軸補間データ
Dim Data4Cip(1) As DATA_4CIP_MC8500P
Data4Cip(0).EndP1 = 10000
Data4Cip(0).EndP2 = 20000
Data4Cip(0).EndP3 = 30000
Data4Cip(0).EndP4 = 40000
Data4Cip(0).Speed = 0

Data4Cip(1).EndP1 = 2000
Data4Cip(1).EndP2 = -1000
Data4Cip(1).EndP3 = 3000
Data4Cip(1).EndP4 = 4000
Data4Cip(1).Speed = 0

Call Nmc_IPMode(No, IcNo, &HF)           '補間モード設定。X, Y, Z, U軸指定。

' 速度関連パラメータ設定
Call Nmc_StartSpd(No, IcNo, AXIS_X, 4000000) ' 定速ドライブにするため4M
                                              ' 他の設定記述は省略

' 4軸連続補間実行。データ数2, X, Y, Z, U軸
Ret = Nmc_4CIPExecMC8500P_BG(Handle.ToInt32, No, IcNo, Data4Cip, 2, &HF, False)
If Ret = CIP_START Then ' 戻り値正常(補間開始)
    Call MsgBox("補間開始")
End If
End Sub

' WM_CIP_ENDメッセージ受信関数
Protected Overrides Sub WndProc(ByRef m As Message)
    Select Case (m.Msg)
        Case WM_CIP_END ' 連続補間終了メッセージ
            If m.LParam.ToInt32 = CIP_END Then ' 戻り値正常(補間終了)
                Call MsgBox("補間正常終了")
            End If
    End Select
End Sub
```

```

        End If
        Exit Select
    End Select
    MyBase.WndProc(m)
End Sub

[C#]
DATA_4CIP_MC8500P [] Data4Cip = new DATA_4CIP_MC8500P[2];    // 4軸連続補間データ用
// 補間データ設定
Data4Cip[0].EndP1 = 1000;
Data4Cip[0].EndP2 = 2000;
Data4Cip[0].EndP3 = 3000;
Data4Cip[0].EndP4 = 3000;
Data4Cip[0].Speed = 0;

Data4Cip[1].EndP1 = 2000;
Data4Cip[1].EndP2 = -1000;
Data4Cip[1].EndP3 = 3000;
Data4Cip[1].EndP4 = -2000;
Data4Cip[1].Speed = 0;

MC8000P.Nmc_IPMode(No, IcNo, 0x000F);    //補間モード設定。X, Y, Z, U軸指定。

// この関数は補間が終了するまで制御が戻りません
// 4軸連続補間実行
// バックグラウンドで実行する
Ret = MC8000P.Nmc_4CIPExec_BG((System.IntPtr)parent.Handle, gBoardNo, int IcNo,
    Data4Cip, 2, 0xF, SpdChgFlg, ContinueFlg);

if(Ret == Nmc_Status.CIP_START)    // 連続補間処理正常開始

//WM_BP_ENDメッセージ受信関数
protected override void WndProc(ref Message m)
{
    // 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
    base.WndProc ( ref m );
    if ( m.Msg == (int)MSG_ID.WM_CIP_END )
    {
        if((uint)m.LParam == Nmc_Status.CIP_END)    // 連続補間処理正常終了
        }
    }
}

```

注意

この関数を実行する前に、必ず補間モード設定(2A)Hで補間軸を設定して下さい。補間モード設定で設定した軸と、引数IpAxisで指定する軸は必ず同じ値にして下さい。
この関数は定速ドライブで動作します。定速ドライブになるように速度パラメータを設定してください。

関数名	機能 及び 内容
Nmc_IPStop	<p>補間処理を中断する。 補間ドライブを即停止し、Nmc_xxxの補間関数で実行中の補間処理を終了します。</p> <p>Nmc_IPStopを使用して補間処理を中断した場合、各補間関数の戻り値は下記のエラーコードです。 ◆BP補間 : BP_USER_STOP ◆連続補間 : CIP_USER_STOP</p> <p>VC BOOL Nmc_IPStop(int No, int IcNo); VB.NET Function Nmc_IPStop(ByVal No As Integer, ByVal IcNo As Integer) As Integer C# bool MC8000P.Nmc_IPStop(int No, int IcNo);</p> <p>入力パラメータ No ボード番号 (ボード上のロータリースイッチの値(0~15)) IcNo IC番号(0~1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0, IC-Bが1。 詳細は「5.1.3 補足説明」(7)参照。</p> <p>戻り値 [VC] 補間処理の中断に成功するとTRUE、失敗するとFALSE [VB.NET] 補間処理の中断に成功すると0以外、失敗すると0 [C#] 補間処理の中断に成功するとtrue、失敗するとfalse</p> <p>使用例 [VC] Nmc_IPStop(No, IcNo); // 補間処理を中断する [VB.NET] Call Nmc_IPStop(No, IcNo) ' 補間処理を中断する [C#] MC8000P.Nmc_IPStop(No, IcNo);</p>
Nmc_IPGetMsgNo	<p>補間終了時に受信した下記メッセージの引数wParamからボード番号とIC番号を取得する。 ◆BP補間 : WM_BP_END ◆連続補間 : WM_CIP_END</p> <p>VC void Nmc_IPGetMsgNo(int wParam, int* No, int* IcNo); VB.NET Sub Nmc_IPGetMsgNo(ByVal wParam As Integer, ByRef No As Integer, ByRef IcNo As Integer) C# bool MC8000P.Nmc_IPGetMsgNo(int wParam, out int No, out int IcNo)</p> <p>入力パラメータ wParam 補間終了時に受信したメッセージの引数wParam No [VC] ボード番号を格納する変数のアドレス。 [VB.NET][C#] ボード番号を格納する変数。 IcNo [VC] IC番号を格納する変数のアドレス。 [VB.NET][C#] IC番号を格納する変数。</p> <p>戻り値 なし</p> <p>使用例 [VC] int BdNo; // ボード番号 int IcNo; // IC番号 Nmc_IPGetMsgNo(wParam, &BdNo, &IcNo); [VB.NET] Dim BdNo As Integer ' ボード番号 Dim IcNo As Integer ' IC番号 (0~1) Call Nmc_IPGetMsgNo(m.WParam.ToInt32(), BdNo, IcNo) [C#] int BdNo; // ボード番号 int IcNo; // IC番号 (0~1) MC8000P.Nmc_IPGetMsgNo(WParam, out BdNo, out IcNo)</p>


```

DataHL.Speed = 1000          ' 速度
DataHL.CenterX = 0          ' X軸円弧中心点
DataHL.CenterY = 10000     ' Y軸円弧中心点
DataHL.EndPX = 0           ' X軸終点
DataHL.EndPY = 0           ' Y軸終点
DataHL.MoveZ = 3000        ' Z軸移動量
DataHL.MoveU = 400         ' U軸移動量
DataHL.HINum = 1           ' ヘリカル回転数
DataHL.HIValue = 0         ' ヘリカル演算値
DataHL.IpMode = IpMode     ' 補間モード設定

' ヘリカル演算実行
Res = Nmc_HLValueExec(0, 0, DataHL)
If Res = HL_VALUE_END Then ' ヘリカル演算正常終了
    ' ヘリカル演算値取得
    Value = DataHL.HIValue ' 演算終了後、演算結果がHIValueに格納される
    Call MsgBox("演算終了")
End If

```

[C#]

```

DATA_HL DataHL = new DATA_HL();

// ヘリカル補間データ設定
DataHL.Direction = (ushort)HL_DIR.HL_DIR_CCW; // 回転方向(CCW円弧補間)
DataHL.Speed = 1000; // 速度
DataHL.CenterX = 0; // X軸円弧中心点
DataHL.CenterY = 10000; // Y軸円弧中心点
DataHL.EndPX = 0; // X軸終点
DataHL.EndPY = 0; // Y軸終点
DataHL.MoveZ = 3000; // Z軸移動量
DataHL.MoveU = 400; // U軸移動量
DataHL.HINum = 1; // ヘリカル回転数
DataHL.HIValue = 0; // ヘリカル演算値

// 補間モード設定
DataHL.IpMode.Cxiv = false; // 補間軸入れ替え無し
DataHL.IpMode.Lmdf = true; // 短軸パルス均一化有効
DataHL.IpMode.Vcnst = 3; // 2軸高精度線速一定

Ret = MC8000P.Nmc_HLValueExec(0, 0, ref DataHL);

if (Ret == Nmc_Status.HL_VALUE_END) // ヘリカル演算正常終了
{
    // ヘリカル演算値取得
    Value = DataHL.HIValue; // 演算終了後、演算結果がHIValueに格納される
}

```

注意

本関数を実行すると、ドライバ内で、引数のヘリカル補間データの構造体の値で補間モード設定(2A)Hを行います。関数実行前に設定した補間モード設定(2A)Hは無効となりますのでご注意ください。
本関数を実行中に、同じICに対して本関数をコールしないで下さい。

[VB.NET]

```

Dim IpMode As DATA_IPMODE ' 補間モード
Dim DataHL As DATA_HL ' ヘリカル補間データ

' 補間モード設定
IpMode.Cxiv = False ' 円弧補間のときの、補間軸の入れ替えなし
IpMode.Lmdf = False ' 短軸パルス均一化有効
IpMode.Vcnst = 3 ' 2軸高精度線速一定

' ヘリカル補間データ設定
DataHL.Direction = HL_DIR_CCW ' 回転方向 (CCW円弧補間)
DataHL.Speed = 1000 ' 速度
DataHL.CenterX = 0 ' X軸円弧中心点
DataHL.CenterY = 10000 ' Y軸円弧中心点
DataHL.EndPX = 0 ' 終点 (X軸)
DataHL.EndPY = 0 ' 終点 (Y軸)
DataHL.MoveZ = 3000 ' Z軸移動量
DataHL.MoveU = 400 ' U軸移動量
DataHL.HINum = 1 ' ヘリカル回転数
DataHL.HIValue = 0 ' ヘリカル演算値 (ヘリカル演算値をドライバ内で計算する)
DataHL.IpMode = IpMode ' 補間モード設定

' ヘリカルドライブ実行
Res = Nmc_HLExec(0, 0, DataHL)
If Res = HL_START Then ' ヘリカル補間開始
    Call MsgBox("ヘリカル補間開始")
End If
' ヘリカル演算値取得
Value = DataHL.HIValue ' 演算結果がHIValueに格納される
    
```

[C#]

```

DATA_HL DataHL = new DATA_HL();

// ヘリカル補間データ設定
DataHL.Direction = (ushort)HL_DIR.HL_DIR_CCW; // 回転方向 (CCW円弧補間)
DataHL.Speed = 1000; // 速度
DataHL.CenterX = 0; // X軸円弧中心点
DataHL.CenterY = 10000; // Y軸円弧中心点
DataHL.EndPX = 0; // X軸終点
DataHL.EndPY = 0; // Y軸終点
DataHL.MoveZ = 3000; // Z軸移動量
DataHL.MoveU = 400; // U軸移動量
DataHL.HINum = 1; // ヘリカル回転数
DataHL.HIValue = 0; // ヘリカル演算値 (ヘリカル演算値をドライバ内で計算する)

// 補間モード設定
DataHL.IpMode.Cxiv = false; // 補間軸入れ替え無し
DataHL.IpMode.Lmdf = true; // 短軸パルス均一化有効
DataHL.IpMode.Vcnst = 3; // 2軸高精度線速一定

Ret = MC8000P.Nmc_HLValueExec(0, 0, ref DataHL);

if (Ret == Nmc_Status.HL_START) // ヘリカル補間開始
{
    // ヘリカル演算値取得
    Value = DataHL.HIValue; // 演算終了後、演算結果がHIValueに格納される
}
    
```

注意

本関数を実行すると、ドライバ内で、引数のヘリカル補間データの構造体の値で補間モード設定 (2A) Hを行います。関数実行前に設定した補間モード設定 (2A) Hは無効となりますのでご注意ください。
この関数は定速ドライブで動作します。そのため、関数内でX軸の初速度はドライブ速度の値に設定されます。
本関数を実行中に、同じICに対して本関数をコールしないで下さい。

5.1.3 補足説明

(1) 各定義は、下記のファイルで行っています。

```
VC++   ・・・MC8000P_DLL.h
VB.NET ・・・MC8000P_DLL.vb
C#     ・・・入力支援により各定義を参照、引用できます。
```

VC++、VB.NET、C#の定義内容を以下に示します。

①レジスタ番号

[VC]

```
#define MCX_WR0      0x0000 // WR 0
#define MCX_WR1      0x0001 // WR 1
#define MCX_WR2      0x0002 // WR 2
#define MCX_WR3      0x0003 // WR 3
#define MCX_WR4      0x0004 // WR 4
#define MCX_WR5      0x0005 // WR 5
#define MCX_WR6      0x0006 // WR 6
#define MCX_WR7      0x0007 // WR 7

#define MCX_RR0      0x0000 // RR 0
#define MCX_RR1      0x0001 // RR 1
#define MCX_RR2      0x0002 // RR 2
#define MCX_RR3      0x0003 // RR 3
#define MCX_RR4      0x0004 // RR 4
#define MCX_RR5      0x0005 // RR 5
#define MCX_RR6      0x0006 // RR 6
#define MCX_RR7      0x0007 // RR 7
```

[C#]

```
// ■Nmc_OutPort/Nmc_InPort用
// 搭載ICが1つのとき WR0_A ~ WR7_A/RR0_A ~ RR7_Aを使用
// 搭載ICが2つのとき IC_AにはWR0_A ~ WR7_A/RR0_A ~ RR7_Aを使用
//                      IC_BにはWR0_B ~ WR7_B/RR0_B ~ RR7_Bを使用
// ■Nmc_WriteReg/Nmc_ReadReg用
// WR0 ~ WR7/RR0 ~ RR7を使用

public enum REG_MCX : int
{
    //■Nmc_OutPort用
    // ライトレジスタのアドレス
    // IC-AのWR0~WR7
    WR0_A =0x0000,
    WR1_A =0x0001,
    WR2_A =0x0002,
    WR3_A =0x0003,
    WR4_A =0x0004,
    WR5_A =0x0005,
    WR6_A =0x0006,
    WR7_A =0x0007,

    // IC-BのWR0~WR7
    WR0_B =0x0008,
    WR1_B =0x0009,
    WR2_B =0x000A,
    WR3_B =0x000B,
    WR4_B =0x000C,
    WR5_B =0x000D,
    WR6_B =0x000E,
    WR7_B =0x000F,

    //■Nmc_InPort用
    // リードレジスタのアドレス
    // IC-AのRR0~RR7
```

```

RR0_A =0x0000,
RR1_A =0x0001,
RR2_A =0x0002,
RR3_A =0x0003,
RR4_A =0x0004,
RR5_A =0x0005,
RR6_A =0x0006,
RR7_A =0x0007,

// IC-BのRR0~RR7
RR0_B =0x0008,
RR1_B =0x0009,
RR2_B =0x000A,
RR3_B =0x000B,
RR4_B =0x000C,
RR5_B =0x000D,
RR6_B =0x000E,
RR7_B =0x000F,

//■Nmc_WriteReg用
// ライトレジスタのアドレス
WR0   =0x0000,
WR1   =0x0001,
WR2   =0x0002,
WR3   =0x0003,
WR4   =0x0004,
WR5   =0x0005,
WR6   =0x0006,
WR7   =0x0007,

//■Nmc_ReadReg用
// リードレジスタのアドレス
RR0   =0x0000,
RR1   =0x0001,
RR2   =0x0002,
RR3   =0x0003,
RR4   =0x0004,
RR5   =0x0005,
RR6   =0x0006,
RR7   =0x0007
}
(使用例) REG_MCXのWR0の場合は      REG_MCX.WR0

```

②軸定義

[VC]

```

#define AXIS_ALL      0xF // 全軸
#define AXIS_X        0x1 // X軸
#define AXIS_Y        0x2 // Y軸
#define AXIS_Z        0x4 // Z軸
#define AXIS_U        0x8 // U軸
#define AXIS_NONE     0   // 軸指定無し

```

[C#]

```

public enum AXIS : int
{
    // 軸定義
    ALL      =0xF, // 全軸
    X        =0x1, // X軸
    Y        =0x2, // Y軸
    Z        =0x4, // Z軸
    U        =0x8, // U軸
    NONE     =0    // 軸指定無し
}

```

(使用例) 全軸の場合は AXIS.ALL

③デバイスID

[VC]

デバイスIDとは、ボードに割り当てられたの固有の番号です。
各ボードの番号は以下の通りです。

ボード	デバイスID
MC8541P / MC8541Pe	0xA808
MC8581P / MC8581Pe	0xA809

[VC]

```
#define ID_MC8541P      0xA808      // MC8541P及びMC8541Pe
#define ID_MC8581P      0xA809      // MC8581P及びMC8581Pe
```

[C#]

```
public enum Dev_ID : ushort
{
    MC8541P      =0xA808,      // MC8541P及びMC8541Pe
    MC8581P      =0xA809      // MC8581P及びMC8581Pe
}
```

(使用例) MC8541PはDev_ID.MC8541P、MC8581PはDev_ID.MC8581P

④コマンド定義 ※使用できるコマンドはMCX514の取扱説明書参照。

[VC]

```
// ドライブ命令
#define MC8500P_CMD_DRVRL      0x0050      // 相対位置ドライブ
#define MC8500P_CMD_DRVNR      0x0051      // 反相対位置ドライブ
#define MC8500P_CMD_DRVVP      0x0052      // +方向連続パルスドライブ
#define MC8500P_CMD_DRVVM      0x0053      // -方向連続パルスドライブ
#define MC8500P_CMD_DRVAB      0x0054      // 絶対位置ドライブ
#define MC8500P_CMD_DRVSBRK     0x0056      // ドライブ減速停止
#define MC8500P_CMD_DRVFBRK     0x0057      // ドライブ即停止
#define MC8500P_CMD_DIRCP       0x0058      // 方向信号+設定
#define MC8500P_CMD_DIRCM       0x0059      // 方向信号-設定
#define MC8500P_CMD_HMSRC       0x005A      // 自動原点出し実行

// 補間命令
#define MC8500P_CMD_1CIP        0x0060      // 1軸直線補間ドライブ (マルチチップ用) ※MC8581P専用
#define MC8500P_CMD_2CIP        0x0061      // 2軸直線補間ドライブ
#define MC8500P_CMD_3CIP        0x0062      // 3軸直線補間ドライブ
#define MC8500P_CMD_4CIP        0x0063      // 4軸直線補間ドライブ
#define MC8500P_CMD_CIPCW       0x0064      // CW円弧補間ドライブ
#define MC8500P_CMD_CIPCCW      0x0065      // CCW円弧補間ドライブ
#define MC8500P_CMD_BPPIP2      0x0066      // 2軸ビットパターン補間ドライブ
#define MC8500P_CMD_BPPIP3      0x0067      // 3軸ビットパターン補間ドライブ
#define MC8500P_CMD_BPPIP4      0x0068      // 4軸ビットパターン補間ドライブ
#define MC8500P_CMD_HLCW        0x0069      // CWヘリカル補間ドライブ
#define MC8500P_CMD_HLCCW       0x006A      // CCWヘリカル補間ドライブ
#define MC8500P_CMD_HLPCW       0x006B      // CWヘリカル演算
#define MC8500P_CMD_HLPCCW      0x006C      // CCWヘリカル演算
#define MC8500P_CMD_DECEN       0x006D      // 減速有効
#define MC8500P_CMD_DECDIS      0x006E      // 減速無効
#define MC8500P_CMD_CLRINTRPT   0x006F      // 補間割り込みクリア
#define MC8500P_CMD_IPSTEP      0x006F      // 補間ステップ

// その他の命令
#define MC8500P_CMD_VINC        0x0070      // 速度増加
#define MC8500P_CMD_VDEC        0x0071      // 速度減少
#define MC8500P_CMD_TMSTA       0x0073      // タイマー始動
#define MC8500P_CMD_TMSTP       0x0074      // タイマー停止
#define MC8500P_CMD_DHOLD       0x0077      // ドライブ開始ホールド
#define MC8500P_CMD_DFREE       0x0078      // ドライブ開始フリー
#define MC8500P_CMD_R2CLR       0x0079      // エラー・終了ステータスクリア
#define MC8500P_CMD_RR3P0       0x007A      // RR3 ページ0表示
#define MC8500P_CMD_RR3P1       0x007B      // RR3 ページ1表示
#define MC8500P_CMD_TXCLR       0x007C      // 終点最大値クリア
```

```
#define MC8500P_CMD_NOP          0x001F    // NOP
#define MC8500P_CMD_RST          0x00FF    // コマンドリセット
```

[C#]

```
public enum CMD : int
{
    // ドライブ命令
    MC8500P_CMD_DRVRL      = 0x0050,      // 相対位置ドライブ
    MC8500P_CMD_DRVNR     = 0x0051,      // 反相対位置ドライブ
    MC8500P_CMD_DRVVP     = 0x0052,      // +方向連続パルスドライブ
    MC8500P_CMD_DRVVM     = 0x0053,      // -方向連続パルスドライブ
    MC8500P_CMD_DRVAB     = 0x0054,      // 絶対位置ドライブ
    MC8500P_CMD_DRVSBRK   = 0x0056,      // ドライブ減速停止
    MC8500P_CMD_DRVFBRK   = 0x0057,      // ドライブ即停止
    MC8500P_CMD_DIRCP     = 0x0058,      // 方向信号+設定
    MC8500P_CMD_DIRCM     = 0x0059,      // 方向信号-設定
    MC8500P_CMD_HMSRC     = 0x005A,      // 自動原点出し実行

    // 補間命令
    MC8500P_CMD_1CIP      = 0x0060,      // 1軸直線ドライブ(マルチチップ用) ※MC8581P専用
    MC8500P_CMD_2CIP      = 0x0061,      // 2軸直線補間ドライブ
    MC8500P_CMD_3CIP      = 0x0062,      // 3軸直線補間ドライブ
    MC8500P_CMD_4CIP      = 0x0063,      // 4軸直線補間ドライブ
    MC8500P_CMD_CIPCW     = 0x0064,      // CW円弧補間ドライブ
    MC8500P_CMD_CIPCCW    = 0x0065,      // CCW円弧補間ドライブ
    MC8500P_CMD_BPIP2     = 0x0066,      // 2軸ビットパターン補間ドライブ
    MC8500P_CMD_BPIP3     = 0x0067,      // 3軸ビットパターン補間ドライブ
    MC8500P_CMD_BPIP4     = 0x0068,      // 4軸ビットパターン補間ドライブ
    MC8500P_CMD_HLCW      = 0x0069,      // CWヘリカル補間ドライブ
    MC8500P_CMD_HLCCW     = 0x006A,      // CCWヘリカル補間ドライブ
    MC8500P_CMD_HLPCW     = 0x006B,      // CWヘリカル演算
    MC8500P_CMD_HLPCCW    = 0x006C,      // CCWヘリカル演算
    MC8500P_CMD_DECEN     = 0x006D,      // 減速有効
    MC8500P_CMD_DECDIS    = 0x006E,      // 減速無効
    MC8500P_CMD_CLRINTRPT = 0x006F,      // 補間割り込みクリア
    MC8500P_CMD_IPSTEP    = 0x006F,      // 補間ステップ

    // その他の命令
    MC8500P_CMD_VINC      = 0x0070,      // 速度増加
    MC8500P_CMD_VDEC      = 0x0071,      // 速度減少
    MC8500P_CMD_DCC       = 0x0072,      // 偏差カウンタクリア出力
    MC8500P_CMD_TMSTA     = 0x0073,      // タイマー始動
    MC8500P_CMD_TMSTP     = 0x0074,      // タイマー停止
    MC8500P_CMD_SPSTA     = 0x0075,      // スプリットパルス開始
    MC8500P_CMD_SPSTP     = 0x0076,      // スプリットパルス停止
    MC8500P_CMD_DHOLD     = 0x0077,      // ドライブ開始ホールド
    MC8500P_CMD_DFREE     = 0x0078,      // ドライブ開始フリー
    MC8500P_CMD_R2CLR     = 0x0079,      // エラー・終了ステータスクリア
    MC8500P_CMD_RR3P0     = 0x007A,      // RR3 ページ0表示
    MC8500P_CMD_RR3P1     = 0x007B,      // RR3 ページ1表示
    MC8500P_CMD_TXCLR     = 0x007C,      // 終点最大値クリア
    MC8500P_CMD_NOP       = 0x001F,      // NOP
    MC8500P_CMD_RST       = 0x00FF,      // コマンドリセット
}
```

(使用例) 2軸直線補間ドライブを指定する場合は CMD.MC8500P_CMD_2CIP

⑤補間終了メッセージ、終了ステータス

[VC]

```
// 補間終了メッセージ
#define WM_BP_END          (WM_USER + 1) // B P 補間終了メッセージ
#define WM_CIP_END        (WM_USER + 2) // 連続補間終了メッセージ

//***** BP補間 終了ステータス *****
// ■正常
```

```

#define BP_START          0x101 // バックグラウンドでBP補間を開始した
#define BP_END           0x102 // BP補間正常終了

// ■補間開始前のエラー
#define BP_CNT_ERR       0x111 // 指定されたデータ数が範囲外
#define BP_ALREADY_EXEC  0x112 // 既にB P補間、あるいは連続補間が実行中
#define BP_THREAD_ERR    0x113 // スレッドを起動できなかった
#define BP_MALLOC_ERR    0x114 // メモリを確保できなかった
#define BP_PARAM_ERR     0x116 // 引数の値が正しくない
#define BP_NOT_OPEN_ERR  0x117 // 指定したボードがオープンされていない
#define BP_OTHER_ERR     0x118 // その他のエラー
#define BP_FUNC_ERR      0x119 // 使用できない関数を使用

// ■補間実行中のエラー
#define BP_STOP          0x121 // BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった場合)
#define BP_USER_STOP    0x122 // BP補間実行中にユーザーが中断した
#define BP_DRIVE_ERR    0x123 // BP補間実行中にボードでエラー発生 (RR0にエラー情報がセットされた)
#define BP_STOP_CERR    0x124 // 連続補間が途中で停止した (RR2エラーによる停止の場合)
#define BP_GETSC_ERR    0x125 // スタックカウンタ読み出しエラー

//***** 連続補間 終了ステータス *****
// ■正常
#define CIP_START        0x201 // バックグラウンドで連続補間を開始した
#define CIP_END          0x202 // 連続補間正常終了

// ■補間開始前のエラー
#define CIP_CNT_ERR      0x211 // 指定されたデータ数が範囲外
#define CIP_ALREADY_EXEC 0x212 // 既にB P補間、あるいは連続補間が実行中
#define CIP_THREAD_ERR   0x213 // スレッドを起動できなかった
#define CIP_MALLOC_ERR   0x214 // メモリを確保できなかった
#define CIP_CMD_ERR      0x215 // コマンドエラー (ユーザーが指定したコマンドが間違っている)
#define CIP_PARAM_ERR    0x216 // 引数の値が正しくない
#define CIP_NOT_OPEN_ERR 0x217 // 指定したボードがオープンされていない
#define CIP_OTHER_ERR    0x218 // その他のエラー
#define CIP_FUNC_ERR     0x219 // 使用できない関数を使用

// ■補間実行中のエラー
#define CIP_STOP         0x221 // 連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった場合)
#define CIP_USER_STOP    0x222 // 連続補間実行中にユーザーが中断した
#define CIP_DRIVE_ERR    0x223 // 連続補間実行中にボードでエラー発生 (RR0にエラー情報がセットされた)
#define CIP_STOP_CERR    0x224 // 連続補間が途中で停止した (RR2エラーによる停止の場合)
#define CIP_GETSC_ERR    0x225 // スタックカウンタ読み出しエラー

//***** ヘリカル補間 終了ステータス *****
// ■正常
#define HL_START         0x301 // ヘリカル補間開始
#define HL_VALUE_END     0x302 // ヘリカル演算終了

// ■ヘリカル補間開始前のエラー
#define HL_CNT_ERR       0x311 // 指定されたデータ数が範囲外
#define HL_PARAM_ERR     0x312 // 引数の値が正しくない
#define HL_NOT_OPEN_ERR  0x313 // 指定したボードがオープンされていない
#define HL_FUNC_ERR      0x314 // 使用できない関数を使用
#define HL_OTHER_ERR     0x315 // その他のエラー

```

[C#]

```

public enum MSG_ID : int
{
    // 補間終了メッセージ
    WM_USER          =0x0400,
    WM_BP_END        =WM_USER+1, // (WM_USER + 1) B P補間終了メッセージ
    WM_CIP_END       =WM_USER+2, // (WM_USER + 2) 連続補間終了メッセージ
}

```

(使用例) MSG_IDのWM_BP_ENDの場合は MSG_ID.WM_BP_END

```

public enum Nmc_Status : uint

```

```

{
//***** BP補間 終了ステータス *****
// ■正常
BP_START          = 0x101,          // バックグラウンドでBP補間を開始した
BP_END            = 0x102,          // BP補間正常終了

// ■補間開始前のエラー
BP_CNT_ERR        = 0x111,          // 指定されたデータ数が範囲外
BP_ALREADY_EXEC   = 0x112,          // 既にB P補間、あるいは連続補間が実行中
BP_THREAD_ERR     = 0x113,          // スレッドを起動できなかった
BP_MALLOC_ERR     = 0x114,          // メモリを確保できなかった
BP_PARAM_ERR      = 0x116,          // 引数の値が正しくない
BP_NOT_OPEN_ERR   = 0x117,          // 指定したボードがオープンされていない
BP_OTHER_ERR      = 0x118,          // その他のエラー
BP_FUNC_ERR       = 0x119,          // 使用できない関数を使用

// ■補間実行中のエラー
BP_STOP           = 0x121, // BP補間が途中で停止した（速度が速く次データのスタックが間に合わなかった場合）
BP_USER_STOP      = 0x122, // BP補間実行中にユーザーが中断した
BP_DRIVE_ERR      = 0x123, // BP補間実行中にボードでエラー発生（RR0にエラー情報がセットされた）
BP_STOP_CERR      = 0x124, // 連続補間が途中で停止した（RR2エラーによる停止の場合）
BP_GETSC_ERR      = 0x125, // スタックカウンタ読み出しエラー

//***** 連続補間 終了ステータス *****
// ■正常
CIP_START         = 0x201,          // バックグラウンドで連続補間を開始した
CIP_END           = 0x202,          // 連続補間正常終了

// ■補間開始前のエラー
CIP_CNT_ERR       = 0x211,          // 指定されたデータ数が範囲外
CIP_ALREADY_EXEC  = 0x212,          // 既にB P補間、あるいは連続補間が実行中
CIP_THREAD_ERR    = 0x213,          // スレッドを起動できなかった
CIP_MALLOC_ERR    = 0x214,          // メモリを確保できなかった
CIP_CMD_ERR       = 0x215,          // コマンドエラー（ユーザーが指定したコマンドが間違っている）
CIP_PARAM_ERR     = 0x216,          // 引数の値が正しくない
CIP_NOT_OPEN_ERR  = 0x217,          // 指定したボードがオープンされていない
CIP_OTHER_ERR     = 0x218,          // その他のエラー
CIP_FUNC_ERR      = 0x219,          // 使用できない関数を使用

// ■補間実行中のエラー
CIP_STOP          = 0x221, // 連続補間が途中で停止した（速度が速く次データのセットが間に合わなかった場合）
CIP_USER_STOP     = 0x222, // 連続補間実行中にユーザーが中断した
CIP_DRIVE_ERR     = 0x223, // 連続補間実行中にボードでエラー発生（RR0にエラー情報がセットされた）
CIP_STOP_CERR     = 0x224, // 連続補間が途中で停止した（RR2エラーによる停止の場合）
CIP_GETSC_ERR     = 0x225, // スタックカウンタ読み出しエラー

//***** ヘリカル補間 終了ステータス *****
// ■正常
HL_START          = 0x301,          // ヘリカル補間開始
HL_VALUE_END      = 0x302,          // ヘリカル演算終了

// ■ヘリカル補間開始前のエラー
HL_CNT_ERR        = 0x311,          // 指定されたデータ数が範囲外
HL_PARAM_ERR      = 0x312,          // 引数の値が正しくない
HL_NOT_OPEN_ERR   = 0x313,          // 指定したボードがオープンされていない
HL_FUNC_ERR       = 0x314,          // 使用できない関数を使用（MC8500PでMC8000P用関数を使用したときなど）
HL_OTHER_ERR      = 0x315,          // その他のエラー
}

```

⑥ヘリカル補間回転方向

```
[VC]
#define HL_DIR_CW          0          // CWヘリカル補間
#define HL_DIR_CCW        1          // CCWヘリカル補間

[C#]
public enum HL_DIR : int
{
    HL_DIR_CW          = 0,          // CWヘリカル補間
    HL_DIR_CCW        = 1          // CCWヘリカル補間
}
```

(2) 軸指定の方法は、下記の通りです。

軸	VC、VB.NET
X	AXIS_X
Y	AXIS_Y
Z	AXIS_Z
U	AXIS_U
全軸	AXIS_ALL

① 1 軸指定の場合

AXIS_X, AXIS_Y, AXIS_Z, AXIS_U のいずれかを指定します。

(使用例) X軸にドライブ速度 1000 を設定する

```
[VC]      Nmc_Speed(No, IcNo, AXIS_X, 1000);
[VB.NET]  Call Nmc_Speed(No, IcNo, AXIS_X, 1000)
[C#]      MC8000P.Nmc_Speed(No, IcNo, AXIS.X, 1000);
```

② 2 軸指定の場合

ビットOR演算子を使用します。

例えばX軸とY軸を一度に指定する場合、

```
[VC] . . . . AXIS_X | AXIS_Y を指定します。
[VB.NET] . . . . AXIS_X Or AXIS_Y を指定します。
[C#] . . . . AXIS.X | AXIS.Y を指定します。
```

(使用例) X, Y軸にドライブ速度 1000 を設定する

```
[VC]      Nmc_Speed(No, IcNo, AXIS_X | AXIS_Y, 1000);
[VB.NET]  Call Nmc_Speed(No, IcNo, AXIS_X Or AXIS_Y, 1000)
[C#]      MC8000P.Nmc_Speed(No, IcNo, AXIS.X | AXIS.Y, 1000);
```

③ 3 軸指定の場合

ビットOR演算子を使用します。

例えばX軸とY軸とZ軸を一度に指定する場合、

```
[VC] . . . . AXIS_X | AXIS_Y | AXIS_Z を指定します。
[VB.NET] . . . . AXIS_X Or AXIS_Y Or AXIS_Z を指定します。
[C#] . . . . AXIS.X | AXIS.Y | AXIS.Z を指定します。
```

(使用例) X, Y, Z軸にドライブ速度 1000 を設定する

```
[VC]      Nmc_Speed(No, IcNo, AXIS_X | AXIS_Y | AXIS_Z, 1000);
[VB.NET]  Call Nmc_Speed(No, IcNo, AXIS_X Or AXIS_Y Or AXIS_Z, 1000)
[C#]      MC8000P.Nmc_Speed(No, IcNo, AXIS.X | AXIS.Y | AXIS.Z, 1000);
```

④ 全軸指定の場合

AXIS_ALL を指定します。

(使用例) 全軸にドライブ速度 1000 を設定する

```
[VC]      Nmc_Speed(No, IcNo, AXIS_ALL, 1000);
[VB.NET]  Call Nmc_Speed(No, IcNo, AXIS_ALL, 1000)
[C#]      MC8000P.Nmc_Speed(No, IcNo, AXIS.ALL, 1000);
```

(3) 補間関数で使用する構造体の定義は、下記の通りです。

①V C

```
// 2軸BP補間
typedef struct _DATA_2BP
{
    USHORT Bp1p;    // BP1Pデータ
    USHORT Bp1m;    // BP1Mデータ
    USHORT Bp2p;    // BP2Pデータ
    USHORT Bp2m;    // BP2Mデータ
} DATA_2BP;

// 3軸BP補間
typedef struct _DATA_3BP
{
    USHORT Bp1p;    // BP1Pデータ
    USHORT Bp1m;    // BP1Mデータ
    USHORT Bp2p;    // BP2Pデータ
    USHORT Bp2m;    // BP2Mデータ
    USHORT Bp3p;    // BP3Pデータ
    USHORT Bp3m;    // BP3Mデータ
} DATA_3BP;

// 4軸BP補間
typedef struct _DATA_4BP
{
    USHORT Bp1p;    // BP1Pデータ
    USHORT Bp1m;    // BP1Mデータ
    USHORT Bp2p;    // BP2Pデータ
    USHORT Bp2m;    // BP2Mデータ
    USHORT Bp3p;    // BP3Pデータ
    USHORT Bp3m;    // BP3Mデータ
    USHORT Bp4p;    // BP4Pデータ
    USHORT Bp4m;    // BP4Mデータ
} DATA_4BP;

// 2軸連続補間
typedef struct _DATA_2CIP_MC8500P
{
    USHORT Command; // 命令番号
                    // (MC8500P_CMD_2CIP, MC8500P_CMD_CIPCW, MC8500P_CMD_CIPCCWのいずれかをセットする)
    long Speed;     // 速度 (速度を変更する場合は1~4, 000, 000、変更しない場合は0をセットする)
    long EndP1;     // 終点 (第1軸)
    long EndP2;     // 終点 (第2軸)
    long Center1;   // 円弧中心点 (第1軸)
    long Center2;   // 円弧中心点 (第2軸)
} DATA_2CIP_MC8500P; // 注: 補間軸は補間モード設定(2A)Hで指定する。

// 3軸連続補間
typedef struct _DATA_3CIP_MC8500P
{
    long EndP1;     // 終点 (第1軸)
    long EndP2;     // 終点 (第2軸)
    long EndP3;     // 終点 (第3軸)
    long Speed;     // 速度 (速度を変更する場合は1~4, 000, 000、変更しない場合は0をセットする)
} DATA_3CIP_MC8500P; // 注: 補間軸は補間モード設定(2A)Hで指定する。
```

```

// 4軸連続補間
typedef struct _DATA_4CIP_MC8500P
{
    long EndP1; // 終点 (第1軸)
    long EndP2; // 終点 (第2軸)
    long EndP3; // 終点 (第3軸)
    long EndP4; // 終点 (第4軸)
    long Speed; // 速度 (速度を変更する場合は1~4, 000, 000、変更しない場合は0をセットする)
} DATA_4CIP_MC8500P; // 注: 補間軸は補間モード設定(2A)Hで指定する。

// 補間モード設定
typedef struct _DATA_IPMODE
{
    BOOL Cxiv; // 円弧補間のときの、補間軸の入れ替え (TURE:入れ替える、FALSE:入れ替えない)
    BOOL Lmdf; // 短軸パルス均一化 (TRUE:有効、FALSE:無効)
    USHORT Vcnst; // 線速一定 (0:線速一定なし、1:2軸簡易、2:3軸簡易、3:2軸高精度)
}
DATA_IPMODE;

// ヘリカル補間
typedef struct _DATA_HL
{
    USHORT Direction; // 回転方向 (HL_DIR_CW, HL_DIR_CCWのいずれかをセットする)
    long Speed; // 速度 (1~2, 000, 000)
    long CenterX; // 円弧中心点 (X軸)
    long CenterY; // 円弧中心点 (Y軸)
    long EndPX; // 終点 (X軸)
    long EndPY; // 終点 (Y軸)
    long MoveZ; // 移動量 (Z軸) (0:Z軸の使用なし、0以外:Z軸の移動量)
    long MoveU; // 移動量 (U軸) (0:U軸の使用なし、0以外:U軸の移動量)
    USHORT HlNum; // ヘリカル回転数 (0~65, 535)
    long HlValue; // ヘリカル演算値
    // (0:ヘリカル演算値を求めてから補間を実行する, 0以上:セットした演算値で補間を実行する)
    DATA_IPMODE IpMode; // 補間モード設定
}
DATA_HL;

```

② V B . N E T

' 2軸BP補間

```

Structure DATA_2BP
    Dim Bp1p As Short ' BP1Pデータ
    Dim Bp1m As Short ' BP1Mデータ
    Dim Bp2p As Short ' BP2Pデータ
    Dim Bp2m As Short ' BP2Mデータ
End Structure

```

' 3軸BP補間

```

Structure DATA_3BP
    Dim Bp1p As Short ' BP1Pデータ
    Dim Bp1m As Short ' BP1Mデータ
    Dim Bp2p As Short ' BP2Pデータ
    Dim Bp2m As Short ' BP2Mデータ
    Dim Bp3p As Short ' BP3Pデータ
    Dim Bp3m As Short ' BP3Mデータ
End Structure

```

```

' 4軸BP補間
Structure DATA_4BP
    Dim Bp1p As Short      ' BP1Pデータ
    Dim Bp1m As Short      ' BP1Mデータ
    Dim Bp2p As Short      ' BP2Pデータ
    Dim Bp2m As Short      ' BP2Mデータ
    Dim Bp3p As Short      ' BP3Pデータ
    Dim Bp3m As Short      ' BP3Mデータ
    Dim Bp4p As Short      ' BP4Pデータ
    Dim Bp4m As Short      ' BP4Mデータ
End Structure

' 2軸連続補間
Structure DATA_2CIP_MC8500P
    Dim Cmd As Short      ' 命令番号 (CMD_IP_2ST, CMD_IP_CW, CMD_IP_CCWのいずれかをセットする)
    Dim Speed As Integer  ' 速度 (速度を変更する場合は1~4, 000, 000、変更しない場合は0をセットする)
    Dim EndP1 As Integer  ' 終点 (第1軸)
    Dim EndP2 As Integer  ' 終点 (第2軸)
    Dim Center1 As Integer ' 円弧中心点 (第1軸)
    Dim Center2 As Integer ' 円弧中心点 (第2軸)
End Structure      ' 注: 補間軸は補間モード設定(2A)Hで指定する

' 3軸連続補間
Structure DATA_3CIP_MC8500P
    Dim EndP1 As Integer  ' 終点 (第1軸)
    Dim EndP2 As Integer  ' 終点 (第2軸)
    Dim EndP3 As Integer  ' 終点 (第3軸)
    Dim Speed As Integer  ' 速度 (速度を変更する場合は1~4, 000, 000、変更しない場合は0をセットする)
End Structure      ' 注: 補間軸は補間モード設定(2A)Hで指定する

' 4軸連続補間
Structure DATA_4CIP_MC8500P
    Dim EndP1 As Integer  ' 終点 (第1軸)
    Dim EndP2 As Integer  ' 終点 (第2軸)
    Dim EndP3 As Integer  ' 終点 (第3軸)
    Dim EndP4 As Integer  ' 終点 (第4軸)
    Dim Speed As Integer  ' 速度 (速度を変更する場合は1~4, 000, 000、変更しない場合は0をセットする)
End Structure      ' 注: 補間軸は補間モード設定(2A)Hで指定する

' 補間モード設定
Structure DATA_IPMODE
    Dim Cxiv As Boolean   ' 円弧補間のときの、補間軸の入れ替え(True:入れ替える、False:入れ替えない)
    Dim Lmdf As Boolean   ' 短軸パルス均一化 (TRUE:有効、FALSE:無効)
    Dim Vcnst As UShort  ' 線速一定 (0:線速一定なし、1:2軸簡易、2:3軸簡易、3:2軸高精度)
End Structure

' ヘリカル補間
Structure DATA_HL
    Dim Direction As UShort ' 回転方向 (HL_DIR_CW: CW円弧補間, HL_DIR_CCW: CCW円弧補間のいずれかをセットする)
    Dim Speed As Integer    ' 速度 (1~2000000)
    Dim CenterX As Integer  ' 円弧中心点 (X軸)
    Dim CenterY As Integer  ' 円弧中心点 (Y軸)
    Dim EndPX As Integer    ' 終点 (X軸)
    Dim EndPY As Integer    ' 終点 (Y軸)
    Dim MoveZ As Integer    ' 移動量 (Z軸) (0:Z軸の使用なし、0以外:Z軸の移動量)
    Dim MoveU As Integer    ' 移動量 (U軸) (0:U軸の使用なし、0以外:U軸の移動量)
    Dim HlNum As UShort     ' ヘリカル回転数 (0~65,535)
    Dim HlValue As Integer  ' ヘリカル演算値 (0:ヘリカル演算値を求めてから補間を実行する,
                          ' 0以上:セットした演算値で補間を実行する)
    Dim IpMode As DATA_IPMODE ' 補間モード設定
End Structure

```

③ C #

```

// 2軸BP補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_2BP
{
    public DATA_2BP(ushort bp1p, ushort bp1m, ushort bp2p, ushort bp2m)
    {
        this.Bp1p    = bp1p;
        this.Bp1m    = bp1m;
        this.Bp2p    = bp2p;
        this.Bp2m    = bp2m;
    }
    public ushort Bp1p; // BP1Pデータ
    public ushort Bp1m; // BP1Mデータ
    public ushort Bp2p; // BP2Pデータ
    public ushort Bp2m; // BP2Mデータ
}

// 3軸BP補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_3BP
{
    public DATA_3BP(ushort bp1p, ushort bp1m, ushort bp2p, ushort bp2m, ushort bp3p, ushort bp3m)
    {
        this.Bp1p    = bp1p;
        this.Bp1m    = bp1m;
        this.Bp2p    = bp2p;
        this.Bp2m    = bp2m;
        this.Bp3p    = bp3p;
        this.Bp3m    = bp3m;
    }
    public ushort Bp1p; // BP1Pデータ
    public ushort Bp1m; // BP1Mデータ
    public ushort Bp2p; // BP2Pデータ
    public ushort Bp2m; // BP2Mデータ
    public ushort Bp3p; // BP3Pデータ
    public ushort Bp3m; // BP3Mデータ
}

// 4軸BP補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_4BP
{
    public DATA_4BP(ushort bp1p, ushort bp1m, ushort bp2p, ushort bp2m, ushort bp3p, ushort bp4p, ushort bp3m)
    {
        this.Bp1p    = bp1p;
        this.Bp1m    = bp1m;
        this.Bp2p    = bp2p;
        this.Bp2m    = bp2m;
        this.Bp3p    = bp3p;
        this.Bp3m    = bp3m;
        this.Bp4p    = bp4p;
        this.Bp4m    = bp4m;
    }
    public ushort Bp1p; // BP1Pデータ
    public ushort Bp1m; // BP1Mデータ
    public ushort Bp2p; // BP2Pデータ
    public ushort Bp2m; // BP2Mデータ
    public ushort Bp3p; // BP3Pデータ
    public ushort Bp3m; // BP3Mデータ
    public ushort Bp4p; // BP4Pデータ
    public ushort Bp4m; // BP4Mデータ
}

```

```

// 2軸連続補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_2CIP_MC8500P
{
    public DATA_2CIP_MC8500P(ushort command, ushort speed, int endp1, int endp2, int center1, int center2)
    {
        this.Command = command;
        this.Speed = speed;
        this.EndP1 = endp1;
        this.EndP2 = endp2;
        this.Center1 = center1;
        this.Center2 = center2;
    }
    public ushort Command; // 命令番号 (CMD_IP_2ST, CMD_IP_CW, CMD_IP_CCWのいずれかをセットする)
    public ushort Speed; // 速度 (指定する場合は1～4, 000, 000、変更しない場合は0をセットする)
    public int EndP1; // 終点 (第1軸)
    public int EndP2; // 終点 (第2軸)
    public int Center1; // 円弧中心点 (第1軸)
    public int Center2; // 円弧中心点 (第2軸)
} // 注: 補間軸は補間モード設定(2A)Hで指定する

// 3軸連続補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_3CIP_MC8500P
{
    public DATA_3CIP_MC8500P(int endp1, int endp2, int endp3, ushort speed)
    {
        this.EndP1 = endp1;
        this.EndP2 = endp2;
        this.EndP3 = endp3;
        this.Speed = speed;
    }
    public int EndP1; // 終点 (第1軸)
    public int EndP2; // 終点 (第2軸)
    public int EndP3; // 終点 (第3軸)
    public ushort Speed; // 速度 (指定する場合は1～4, 000, 000、変更しない場合は0をセットする)
} // 注: 補間軸は補間モード設定(2A)Hで指定する

// 4軸連続補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_4CIP_MC8500P
{
    public DATA_4CIP_MC8500P(int endp1, int endp2, int endp3, int endp4, ushort speed)
    {
        this.EndP1 = endp1;
        this.EndP2 = endp2;
        this.EndP3 = endp3;
        this.EndP4 = endp4;
        this.Speed = speed;
    }
    public int EndP1; // 終点 (第1軸)
    public int EndP2; // 終点 (第2軸)
    public int EndP3; // 終点 (第3軸)
    public int EndP4; // 終点 (第4軸)
    public ushort Speed; // 速度 (指定する場合は1～4, 000, 000、変更しない場合は0をセットする)
} // 注: 補間軸は補間モード設定(2A)Hで指定する

```

```

// 補間モード設定
[StructLayout(LayoutKind.Sequential)]
public struct DATA_IPMODE
{
    public DATA_IPMODE(bool cxiv, bool lmdf, ushort vcnst)
    {
        this.Cxiv = cxiv;
        this.Lmdf = lmdf;
        this.Vcnst= vcnst;
    }
    public bool    Cxiv;        // 円弧補間のときの、補間軸の入れ替え(TURE:入れ替える、FALSE:入れ替えない)
    public bool    Lmdf        // 短軸パルス均一化(TRUE:有効、FALSE:無効)
    public ushort  Vcnst;      // 線速一定(0:線速一定なし、1:2軸簡易、2:3軸簡易、3:2軸高精度)
}

// ヘリカル補間
[StructLayout(LayoutKind.Sequential)]
public struct DATA_HL
{
    public DATA_HL(ushort direction, int speed, int centerX, int centerY, int endPX, int endPY, int moveZ, int
        moveU, ushort hlNum, int hlValue, DATA_IPMODE ipMode)
    {
        this.Direction =    direction;
        this.Speed =        speed;
        this.CenterX =      centerX;
        this.CenterY =      centerY;
        this.EndPX =        endPX;
        this.EndPY =        endPY;
        this.MoveZ =        moveZ;
        this.MoveU =        moveU;
        this.HlNum =        hlNum;
        this.HlValue =      hlValue;
        this.IpMode =       ipMode;
    }
    public ushort  Direction; // 回転方向 (HL_DIR_CW: C W円弧補間, HL_DIR_CCW : C C W円弧補間のいずれかをセットする)
    public int     Speed;     // 速度 (1 ~ 2 0 0 0 0 0)
    public int     CenterX;   // 円弧中心点 (X軸)
    public int     CenterY;   // 円弧中心点 (Y軸)
    public int     EndPX;     // 終点 (X軸)
    public int     EndPY;     // 終点 (Y軸)
    public int     MoveZ;     // 移動量 (Z軸) (0:Z軸の使用なし、0以外:Z軸の移動量)
    public int     MoveU;     // 移動量 (U軸) (0:U軸の使用なし、0以外:U軸の移動量)
    public ushort  HlNum;     // ヘリカル回転数 (0 ~ 6 5 5 3 5)
    public int     HlValue;   // ヘリカル演算値
    // (0:ヘリカル演算値を求めてから補間を実行する, 0以上:セットした演算値で補間を実行する)
    public DATA_IPMODE IpMode; // 補間モード設定
}

```

構造体の使用例を以下に示します。（V C の場合）

例 1 . 定義、初期化が別の場合

```
DATA_2BP [] Data2Bp = new DATA_2BP[4]; // 2軸BP補間データ

// 補間データ設定
Data2Bp[0].Bp1p = 0xFF30; // 1111 1111 0011 0000 BP1+方向 10パルス
Data2Bp[0].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data2Bp[0].Bp2p = 0; // 0000 0000 0000 0000 BP2+方向 0パルス
Data2Bp[0].Bp2m = 0x84FF; // 1000 0100 1111 1111 BP2-方向 10パルス

Data2Bp[1].Bp1p = 0xAC35; // 1010 1100 0011 0101 BP1+方向 8パルス
Data2Bp[1].Bp1m = 0; // 0000 0000 0000 0000 BP1-方向 0パルス
Data2Bp[1].Bp2p = 0xC000; // 1100 0000 0000 0000 BP2+方向 2パルス
Data2Bp[1].Bp2m = 0x36E7; // 0011 0110 1110 0111 BP2-方向 10パルス

Data2Bp[2].Bp1p = 0x3F3F; // 0011 1111 0011 1111 BP1+方向 12パルス
Data2Bp[2].Bp1m = 0xC000; // 1100 0000 0000 0000 BP1-方向 2パルス
Data2Bp[2].Bp2p = 0xFBDA; // 1111 1011 1101 1010 BP2+方向 12パルス
Data2Bp[2].Bp2m = 0; // 0000 0000 0000 0000 BP2-方向 0パルス

Data2Bp[3].Bp1p = 0; // 0000 0000 0000 0000 BP1+方向 0パルス
Data2Bp[3].Bp1m = 0x1CF2; // 0001 1100 1111 0010 BP1-方向 8パルス
Data2Bp[3].Bp2p = 0xFFFF; // 1111 1111 1111 1111 BP2+方向 16パルス
Data2Bp[3].Bp2m = 0; // 0000 0000 0000 0000 BP2-方向 0パルス
```

例 2 . 定義と初期化を同時に行う場合

```
// 2軸BP補間データ BP1P, BP1M, BP2P, BP2M (MCX514取説3.4.8 ビットパターン補間の实例のデータ)
DATA_2BP[] Data2Bp = new DATA_2BP[]
{
    new DATA_2BP(0xFFE4, 0x0000, 0x0000, 0x03FF),
    new DATA_2BP(0x03FF, 0x4000, 0xFFD0, 0x0000),
    new DATA_2BP(0x0000, 0xFFFF, 0x4AAB, 0x0000),
    new DATA_2BP(0xFE80, 0x000F, 0x1FFF, 0x0000),
    new DATA_2BP(0x97FF, 0x8000, 0x0000, 0x7F20),
};
```

- (4) 補間関数で指定する補間軸(IpAxis)の指定は下記の通りです。
4軸X、Y、Z、Uの中から補間する軸を指定します。

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	0	0	U-EN	Z-EN	Y-EN	X-EN

◆各ビットの説明

D3～0 U-EN～X-EN 補間ドライブを行う軸を指定します。下表にビットに対応した補間軸を示します。
0：補間軸として不使用、1：補間軸として使用

軸	指定ビット
X	D0
Y	D1
Z	D2
U	D3

主軸はX-EN>Y-EN>Z-EN>U-ENの優先順で、1が設定されているビットの軸が選ばれます。

- (5) 連続補間関数実行時の速度変更について

連続補間関数は、補間実行中に速度変更を行う事ができます。各セグメント毎に速度を設定できます。
補間実行中に速度変更を行う場合は、関数のパラメータSpdChgFlgにTRUE(True)を設定します。

◆各セグメント毎の速度の設定方法

DATA_2CIP_MC8500PのSpeed、DATA_3CIP_MC8500PのSpeed、DATA_4CIP_MC8500PのSpeed に、
各セグメントの速度を設定します。

- ・前のセグメントと異なる速度にする場合は、1～4,000,000 の速度を設定します。
- ・前のセグメントと同じ速度のままにする場合は、0 を設定します。

- (6) アドレスの指定

Nmc_OutPort,Nmc_InPort関数のアドレスには、各ボードの取扱説明書に記載しているI/Oアドレスを指定します。

ライトレジスタ、リードレジスタなどのI/Oアドレスは下記の通りです。詳細は各ボードの取扱説明書を参照。

ボード名	ライトレジスタのアドレス	リードレジスタのアドレス
MC8541P / MC8541Pe	0～7	0～7
MC8581P / MC8581Pe	0～F	0～F

注1：アドレスは16進数で記載しています。

注2：ライトレジスタ、リードレジスタにアクセスする場合、Nmc_OutPort,Nmc_InPort関数ではなく、専用の関数を使用した方が便利です。

- (7) IC番号の指定は下記の通りです。

- ① ICを1つ搭載しているボードの場合は0を指定します。
- ② ICを2つ搭載しているボードの場合は次の値を指定します。
 - ・IC-Aの場合は0
 - ・IC-Bの場合は1

ボードの搭載IC数と指定するIC番号は次の通りです。

ボード名	搭載IC数	IC番号
MC8541P / MC8541Pe	1	0
MC8581P / MC8581Pe	2	0, 1

注：ICはMCX514の事を指します。

5.1.4 使用方法

■API関数宣言

API関数宣言は、下記の場所で行っています。

VC++	: MC8000P_DLL.h
VB.NET	: MC8000P_DLL.vb
C#	: 入力支援により各定義を参照、引用できます。

■使用方法

- (1) 開始処理・・・各関数を使用する前にNmc_Openを一度実行して下さい。
- (2) 終了処理・・・プログラム終了時にNmc_Close、又はNmc_CloseAllを実行して下さい。

■ボード番号について

アプリケーションから指定するボード番号は下記の通りです。

	ボードのロータリースイッチの値	アプリケーションから指定するボード番号（10進数）
1	0～9	0～9
2	A～F	10～15

■関数使用時の注意

- (1) VC, VB.NET, C#（全ての言語）について

- ①Nmc_Open関数実行前に各関数を実行した場合の動作保証はできません。
- ②接続していないボードの番号を誤って指定した場合も、各関数の動作の保証はできません。
- ③1枚のボードに対して、2つ以上のアプリケーションから同時にアクセス（オープンなど）を行わないで下さい。
- ④Nmc_Close(Nmc_CloseAll)関数実行後にNmc_Open関数以外の関数を実行した場合の動作保証はできません。

- (2) VC, C#のみ

- ①割り込み処理関数を使用する場合はWindowsの性格上、割り込み発生からユーザー定義ルーチンへ制御が移行するまでの時間を保証することは出来ません。
- ②割り込みを行う場合は、割り込みユーザー関数（Nmc_SetEventで指定した関数）を実行中にクローズ処理（Nmc_Close又はNmc_CloseAll）を実行しないで下さい。
クローズ処理を行う場合は、必ず、割り込みユーザー関数が終了している状態で行って下さい。

■VCにて割り込みを処理する方法

- ①Nmc_Open関数にて割り込みを使用する設定にします。

```
Nmc_Open(No, TRUE); // 第2引数 TRUE : 割り込みを使用する FALSE : 割り込みを使用しない
```

- ②Nmc_SetEvent関数にて割り込みを処理するユーザー関数を設定します。また、許可する割り込みを設定します。

```
Nmc_SetEvent(No, MC_EventProc, lpParam); // ユーザー関数のアドレスと引数を設定  
Nmc_WriteReg1(No, IcNo, AXIS_ALL, 0x0080); // 指定したICの停止時割り込み発生（全軸）
```

- ③割り込みが発生すると、Nmc_SetEvent関数で設定した割り込みユーザー関数が呼び出されます。
割り込みユーザー関数では、割り込み要因を確認します。RR1の割り込み要因は、Nmc_ReadEvent関数にて取得します。

■割り込みユーザー関数例

```
DWORD WINAPI MC_EventProc(LPVOID lpParam)  
{  
    . . . . .  
    long RrIrX, RrIrY, RrIrZ, RrIrU;  
    Nmc_ReadEvent(No, IcNo, &RrIrX, &RrIrY, &RrIrZ, &RrIrU);  
    . . . . .  
    return 0;  
}
```

- ④割込み処理するユーザー関数の設定を解除する場合は Nmc_ResetEvent を実行して下さい。
この関数を実行すると、ボードで割込みが発生してもユーザー関数は呼び出されません。

```
Nmc_ResetEvent(No);
```

■ C#にて割込みを処理する方法 (C#のみ)

- ①MC8000P.Nmc_SetEventにて、割込みを処理するメソッドを設定します。引数は指定できません。
複数枚ボードを使用する場合は、下記のように設定します。

(ボード番号0の場合)

```
MC8000P.callback[0] = new MC8000P.UserThread(isr);           // isrは割込みユーザー関数
bool ret = MC8000P.Nmc_SetEvent(0, MC8000P.callback[0], param); // 第1引数にボード番号0を指定
                                                                // 関数のアドレスと引数を設定
MC8000P.Nmc_WriteReg1(0, (int)IC.A, AXIS.ALL, 0x0080);     // 停止時割込発生(全軸)
. . .
```

(ボード番号1の場合)

```
MC8000P.callback[1] = new MC8000P.UserThread(isr2);        // isrは割込みユーザー関数
bool ret = MC8000P.Nmc_SetEvent(1, MC8000P.callback[1], param); // 第1引数にボード番号1を指定
                                                                // 関数のアドレスと引数を設定
MC8000P.Nmc_WriteReg1(1, (int)IC.A, AXIS.ALL, 0x0080);     // 停止時割込発生(全軸)
. . .
```

- ②割込処理をする関数では、各ボードの割込み要因を確認します。RR1の割込み要因はMC8000P.Nmc_ReadEventにて取得します。

(ボード番号0の割込みユーザー関数)

```
static void isr(int param)
{
    int Rr1X, Rr1Y, Rr1Z, Rr1U;
    MC8000P.Nmc_ReadEvent(0, (int)IC.A, out Rr1X, out Rr1Y, out Rr1Z, out Rr1U);
    . . . . .
}
```

(ボード番号1の割込みユーザー関数)

```
static void isr2(int param)
{
    int Rr1X, Rr1Y, Rr1Z, Rr1U;
    MC8000P.Nmc_ReadEvent(1, (int)IC.A, out Rr1X, out Rr1Y, out Rr1Z, out Rr1U);
    . . . . .
}
```

- ③割込処理をする関数の設定を解除する場合は、MC8000P.Nmc_ResetEventメソッドを使用します。
このメソッドを実行すると、ボードで割込みが発生しても割込みユーザー関数のメソッドは呼び出されません。

■連続補間について

連続補間を行う場合、MCX514取扱説明書の「3.7 連続補間」を必ず参照し、その章に記載している処理をアプリケーションで行ってください。また、連続補間関数（注1）ではこれらの一部の処理をDLLで行っていますので、この連続補間関数を使用して連続補間の処理を行うこともできます。但し、連続補間関数を使用する場合はいくつか注意事項がありますので、ご注意ください。

注1：Nmc_2CIPExecMC8500P, Nmc_3CIPExecMC8500P, Nmc_4CIPExecMC8500P
Nmc_2CIPExecMC8500P_BG, Nmc_3CIPExecMC8500P_BG, Nmc_4CIPExecMC8500P_BG

連続補間関数を使用する場合の注意事項：

短軸パルス均一化は必ず無効に設定してください。有効に設定した場合、予期せぬ動作を起こす可能性があります。連続補間関数では、MCX514のプリバッファに、あらかじめ8セグメント目までのデータをセットしてから実行を開始します。エラーチェックを行い、エラー発生の場合は関数を終了します。エラーが発生していない場合は、RR0の連続補間のプリバッファスタックカウンタ（SC）の確認を行い、可能になった場合は次のセグメントデータや補間命令を書き込みます。このとき、本関数では書き込み可能とするスタックカウンタ（SC）の値を6としています。（本関数は、連続補間開始後のプリバッファは6段までの使用となっています。）本関数は、連続補間が終了するまでこの処理を繰り返します。エラーチェックと次のセグメントデータ書き込み可能チェック処理などがDLL内部で常にループしているため、本関数実行中にアプリケーションで他の処理も行いたい場合は、本関数の使用が適さない場合があります。この場合は、連続補間関数は使用せず、MCX514取扱説明書を参考にしてアプリケーションで連続補間の処理を作成してください。連続補間の加減速ドライブについてはMCX514取扱説明書の「3.8 加減速ドライブでの補間」を参照してください。また、連続補間関数を使用する場合は、関数を実行する前に初速度を400000に設定してください。（本関数実行中に初速度を変更しないでください。）この場合、各セグメント内は定速ドライブになります。

■B P補間（ビットパターン補間）について

B P補間を行う場合、MCX514取扱説明書の「3.4 ビットパターン補間」を必ず参照し、その章に記載している処理をアプリケーションで行ってください。また、B P補間関数（注2）ではこれらの一部の処理をDLLで行っていますので、このB P補間関数を使用してB P補間の処理を行うこともできます。但し、B P補間関数を使用する場合はいくつか注意事項がありますので、ご注意ください。

注2：Nmc_2BPExecMC8500P, Nmc_3BPExecMC8500P, Nmc_4BPExecMC8500P
Nmc_2BPExecMC8500P_BG, Nmc_3BPExecMC8500P_BG, Nmc_4BPExecMC8500P_BG

B P補間関数を使用する場合の注意事項：

B P補間関数では、MCX514のプリバッファに、あらかじめ8セグメント目までのデータをセットしてから実行を開始します。エラーチェックを行い、エラー発生の場合は関数を終了します。エラーが発生していない場合は、RR0の連続補間のプリバッファスタックカウンタ（SC）の確認を行い、可能になった場合は次のセグメントデータや補間命令を書き込みます。このとき、本関数では書き込み可能とするスタックカウンタ（SC）の値を6としています。（本関数は、連続補間開始後のプリバッファは6段までの使用となっています。）本関数は、連続補間が終了するまでこの処理を繰り返します。エラーチェックと次のセグメントデータ書き込み可能チェック処理などがDLL内部で常にループしているため、本関数実行中にアプリケーションで他の処理も行いたい場合は、本関数の使用が適さない場合があります。この場合は、B P補間関数は使用せず、MCX514取扱説明書を参考にしてアプリケーションでB P補間の処理を作成してください。また、B P補間の加減速ドライブについては、MCX514取扱説明書の「3.8 加減速ドライブでの補間」を参照してください。

■補間関数使用時の注意

- (1) 下記の補間関数について、1つのICに対し一度に実行できるのは1つの補間関数のみです。補間関数実行中に、別の補間関数は実行できません。実行した場合、エラーが返ります。

Nmc_2BPExecMC8500P	Nmc_2BPExecMC8500P_BG	Nmc_2CIPExecMC8500P	Nmc_2CIPExecMC8500P_BG
Nmc_3BPExecMC8500P	Nmc_3BPExecMC8500P_BG	Nmc_3CIPExecMC8500P	Nmc_3CIPExecMC8500P_BG
Nmc_4BPExecMC8500P	Nmc_4BPExecMC8500P_BG	Nmc_4CIPExecMC8500P	Nmc_4CIPExecMC8500P_BG

- (2) 上記の補間関数実行中は、下記の処理を実行しないで下さい。

- ①補間命令（60h～6Fh）の実行
- ②補間モード設定（2Ah）の変更

- (3) 下記の補間関数はバックグラウンドで実行する為、補間関数開始時に補間データ用のメモリを確保し、ユーザーが指定した補間データをコピーします。そして、バックグラウンドで実行していた補間処理が終了する時にそのメモリを解放し、ユーザーウィンドウにメッセージを送信します。

この為、下記補間関数がバックグラウンドで実行している最中にクローズ処理（Nmc_Close 又は Nmc_CloseAll）を実行しないで下さい。

また、下記補間関数がバックグラウンドで実行している最中にアプリケーションを終了しないで下さい。

補間関数の実行を途中で止めたい時は、補間中断関数(Nmc_IPStop)を実行し、必ず中断メッセージを受け取って下さい。

Nmc_2BPExecMC8500P_BG	Nmc_3BPExecMC8500P_BG	Nmc_4BPExecMC8500P_BG
Nmc_2CIPExecMC8500P_BG	Nmc_3CIPExecMC8500P_BG	Nmc_4CIPExecMC8500P_BG

■ヘリカル補間関連関数使用時の注意

- (1) 下記のヘリカル補間関連関数について、1つのICに対し一度に実行できるのは1つの関数のみです。
 ヘリカル補間関数(Nmc_HLExec)実行後のドライブ中に、ヘリカル演算関数(Nmc_HLValueExe)は実行しないでください。
 この場合、ヘリカル補間ドライブの停止を確認してから、ヘリカル演算関数(Nmc_HLValueExe)を実行してください。
 同様に、ヘリカル演算関数(Nmc_HLValueExe)実行中に、ヘリカル補間関数(Nmc_HLExec)を実行しないで下さい。

Nmc_HLValueExe	Nmc_HLExec
----------------	------------

- (2) 上記のヘリカル補間関連関数実行時、補間モード設定(2Ah)が引数で指定した内容で設定されます。

- (3) ヘリカル補間関数(Nmc_HLExec)は定速ドライブで実行します。
 そのため本関数実行時、X軸の初速度が指定したドライブ速度の値で設定されます。

- (4) 上記のヘリカル補間関連関数実行後の動作中は、下記の処理を実行しないでください。

- ①補間命令(60h~6Fh)の実行
- ②補間モード設定(2Ah)の変更

- (5) 上記のヘリカル補間関連関数実行後の動作中は、下記の連続補間関数は実行しないでください。
 ヘリカル演算、ヘリカル補間の動作終了を確認してから実行してください。

Nmc_2BPExecMC8500P	Nmc_2BPExecMC8500P_BG	Nmc_2CIPExecMC8500P	Nmc_2CIPExecMC8500P_BG
Nmc_3BPExecMC8500P	Nmc_3BPExecMC8500P_BG	Nmc_3CIPExecMC8500P	Nmc_3CIPExecMC8500P_BG
Nmc_4BPExecMC8500P	Nmc_4BPExecMC8500P_BG	Nmc_4CIPExecMC8500P	Nmc_4CIPExecMC8500P_BG

■マルチスレッドアプリケーション開発時の注意

ここでは、マルチスレッドで動作するアプリケーションを開発する際の注意事項を説明します。

Nmc_xxx の関数では、軸切り替え処理、WR 6, WR 7にデータを書き込む処理、RR 6, RR 7にデータを読み出す処理を実行している関数があります。それぞれの Nmc_xxx 関数は次の通りです。

◆軸切り替え処理を実行している関数

Nmc_Reset	Nmc_Command	Nmc_Command_IP			
Nmc_WriteReg0	Nmc_WriteReg1	Nmc_WriteReg2	Nmc_WriteReg3		
Nmc_ReadReg2	Nmc_ReadReg3P	Nmc_ReadReg3			
Nmc_Jerk	Nmc_DJerk	Nmc_Acc	Nmc_Dec	Nmc_StartSpd	Nmc_Speed
Nmc_Pulse	Nmc_Pulse__VB	Nmc_DecP	Nmc_DecP__VB	Nmc_Center	Nmc_Lp
Nmc_Ep	Nmc_CompP	Nmc_CompM	Nmc_AccOfst	Nmc_HomeSpd	Nmc_LpMax
Nmc_RpMax	Nmc_MR0	Nmc_MR1	Nmc_MR2	Nmc_MR3	
Nmc_SpeedInc	Nmc_Timer				
Nmc_MRmMode	Nmc_PIO1Mode	Nmc_PIO2Mode	Nmc_HMSrch1Mode	Nmc_HMSrch2Mode	
Nmc_FilterMode	Nmc_Sync0Mode	Nmc_Sync1Mode	Nmc_Sync2Mode	Nmc_Sync3Mode	
Nmc_ReadLp	Nmc_ReadEp	Nmc_ReadSpeed	Nmc_ReadAccDec	Nmc_ReadMR0	
Nmc_ReadMR1	Nmc_ReadMR2	Nmc_ReadMR3	Nmc_ReadCT	Nmc_ReadWR1	Nmc_ReadWR2
Nmc_ReadWR3	Nmc_ReadMRM	Nmc_ReadP1M	Nmc_ReadP2M		
Nmc_ReadAc	Nmc_ReadStartSpd	Nmc_ReadSetSpeed	Nmc_ReadPulse		
Nmc_2BPExecMC8500P	Nmc_3BPExecMC8500P	Nmc_4BPExecMC8500P			
Nmc_2BPExecMC8500P_BG	Nmc_3BPExecMC8500P_BG	Nmc_4BPExecMC8500P_BG			
Nmc_2CIPExecMC8500P	Nmc_3CIPExecMC8500P	Nmc_4CIPExecMC8500P			
Nmc_2CIPExecMC8500P_BG	Nmc_3CIPExecMC8500P_BG	Nmc_4CIPExecMC8500P_BG			
Nmc_HLValueExec	Nmc_HLExec				
Nmc_WriteRegSetAxis	Nmc_ReadRegSetAxis	Nmc_WriteData			
Nmc_ReadData					

◆WR 6, WR 7にデータを書き込む処理を実行している関数

Nmc_Jerk	Nmc_DJerk	Nmc_Acc	Nmc_Dec	Nmc_StartSpd	Nmc_Speed
Nmc_Pulse	Nmc_Pulse__VB	Nmc_DecP	Nmc_DecP__VB	Nmc_Center	Nmc_Lp
Nmc_Ep	Nmc_CompP	Nmc_CompM	Nmc_AccOfst	Nmc_HomeSpd	Nmc_LpMax
Nmc_RpMax	Nmc_MR0	Nmc_MR1	Nmc_MR2	Nmc_MR3	
Nmc_SpeedInc	Nmc_Timer				
Nmc_MRmMode	Nmc_PIO1Mode	Nmc_PIO2Mode	Nmc_HMSrch1Mode	Nmc_HMSrch2Mode	
Nmc_FilterMode	Nmc_Sync0Mode	Nmc_Sync1Mode	Nmc_Sync2Mode	Nmc_Sync3Mode	
Nmc_2BPExecMC8500P	Nmc_3BPExecMC8500P	Nmc_4BPExecMC8500P			
Nmc_2BPExecMC8500P_BG	Nmc_3BPExecMC8500P_BG	Nmc_4BPExecMC8500P_BG			
Nmc_2CIPExecMC8500P	Nmc_3CIPExecMC8500P	Nmc_4CIPExecMC8500P			
Nmc_2CIPExecMC8500P_BG	Nmc_3CIPExecMC8500P_BG	Nmc_4CIPExecMC8500P_BG			
Nmc_HLValueExec	Nmc_HLExec				
Nmc_WriteReg6	Nmc_WriteReg7				
Nmc_WriteRegSetAxis	Nmc_WriteData				

Nmc_OutPortまたはNmc_WriteRegでWR6, WR7に書いた場合

◆RR 6, RR 7にデータを読み出す処理を実行している関数

Nmc_ReadLp	Nmc_ReadEp	Nmc_ReadSpeed	Nmc_ReadAccDec	Nmc_ReadMR0	Nmc_ReadMR1
Nmc_ReadMR2	Nmc_ReadMR3	Nmc_ReadCT	Nmc_ReadTX		
Nmc_ReadCHLN	Nmc_ReadHLV	Nmc_ReadWR1	Nmc_ReadWR2	Nmc_ReadWR3	
Nmc_ReadMRM	Nmc_ReadP1M	Nmc_ReadP2M	Nmc_ReadAc	Nmc_ReadStartSpd	
Nmc_ReadSetSpeed	Nmc_ReadPulse				

Nmc_ReadData

WR 1～WR 3書き込み、RR 2～RR 3読み出し、データ書き込み命令、データ読み出し命令を実行する場合、基本的には次の Nmc_xxx 関数を使用して下さい。

◆WR 1～WR 3書き込み

Nmc_WriteReg1	Nmc_WriteReg2	Nmc_WriteReg3	Nmc_WriteRegSetAxis
---------------	---------------	---------------	---------------------

◆RR 2～RR 3読み出し

Nmc_ReadReg2	Nmc_ReadReg3P	Nmc_ReadReg3	Nmc_ReadRegSetAxis
--------------	---------------	--------------	--------------------

◆データ書き込み命令

Nmc_Jerk	Nmc_DJerk	Nmc_Acc	Nmc_Dec	Nmc_StartSpd
Nmc_Speed	Nmc_Pulse	Nmc_Pulse__VB	Nmc_DecP	Nmc_DecP__VB
Nmc_Center	Nmc_Lp	Nmc_Ep	Nmc_CompP	Nmc_CompM
Nmc_AccOfst	Nmc_HomeSpd	Nmc_LpMax	Nmc_RpMax	Nmc_MR0
Nmc_MR1	Nmc_MR2	Nmc_MR3	Nmc_SpeedInc	Nmc_Timer
Nmc_MRmMode	Nmc_PIO1Mode	Nmc_PIO2Mode	Nmc_HMSrch1Mode	Nmc_HMSrch2Mode
Nmc_FilterMode	Nmc_Sync0Mode	Nmc_Sync1Mode	Nmc_Sync2Mode	Nmc_Sync3Mode

Nmc_WriteData

◆データ読み出し命令

Nmc_ReadLp	Nmc_ReadEp	Nmc_ReadSpeed	Nmc_ReadAccDec	Nmc_ReadMR0
Nmc_ReadMR1	Nmc_ReadMR2	Nmc_ReadMR3	Nmc_ReadCT	Nmc_ReadTX
Nmc_ReadCHLN	Nmc_ReadHLV	Nmc_ReadWR1	Nmc_ReadWR2	Nmc_ReadWR3
Nmc_ReadMRM	Nmc_ReadP1M	Nmc_ReadP2M	Nmc_ReadAc	Nmc_ReadStartSpd
Nmc_ReadSetSpeed	Nmc_ReadPulse			

Nmc_ReadData

WR 1～WR 3 書き込み、RR 2～RR 3 読み出し、データ書き込み命令、データ読み出し命令を実行する際、これらの関数を使用せずに同じ処理を行った場合、マルチスレッド環境では注意が必要です。

(1) 例えば、WR 1 書き込み時には Nmc_WriteReg1 を使用しますが、それ以外の方法としては下記の方法などがあります。

```
① Nmc_OutPort(No, MCX_WR0, 0x011F); // X軸に切り替える (I C-A)
② Nmc_OutPort(No, MCX_WR1, Data); // WR 1 書き込み (I C-A)
```

また、次のような関数でも同じ処理を実行できます。

```
③ Nmc_WriteReg(No, IcNo, MCX_WR0, 0x011F); // X軸に切り替える
④ Nmc_WriteReg(No, IcNo, MCX_WR1, Data); // WR 1 書き込み
```

この場合、①と②の間、③と④の間に、軸を切り替える Nmc_xxx 関数が実行された場合、別の軸のWR 1 にデータが書かれてしまいます。

(2) 例えば、速度設定時には Nmc_Speed を使用しますが、それ以外の方法としては下記の方法などがあります。

```
① Nmc_OutPort( No, MCX_WR6, Data ); // WR 6 書き込み (I C-A)
② Nmc_OutPort( No, MCX_WR0, 0x0105 ); // WR 6 データを X 軸の速度に設定する (I C-A)
```

また、次のような関数でも同じ処理を実行できます。

```
③ Nmc_WriteReg6(No, IcNo, Data); // WR 6 書き込み
④ Nmc_Command(No, IcNo, AXIS_X, 0x05); // WR 6 データを X 軸の速度に設定する
```

この場合、①と②の間、③と④の間に、WR 6, WR 7 にデータを書き込む Nmc_xxx 関数が実行された場合、別のデータが速度に書かれてしまいます。

(3) 例えば、論理位置カウンタ読み出し時には Nmc_ReadLp を使用しますが、それ以外の方法としては下記の方法などがあります。

```
① Nmc_OutPort( No, MCX_WR0, 0x0130 ); // X 軸の論理位置カウンタを RR 6, RR 7 に読み出す (I C-A)
② d6 = Nmc_InPort( No, MCX_RR6 ); // RR 6 を読み出す (I C-A)
③ d7 = Nmc_InPort( No, MCX_RR7 ); // RR 7 を読み出す (I C-A)
```

この場合、①と②の間、②と③の間に、RR 6, RR 7 にデータを読み出す Nmc_xxx 関数が実行された場合、ここでは別のデータが読み出されてしまいます。

このように、マルチスレッド環境では、目的の処理を実行するのに2回以上 API 関数をコールする場合は、そのような処理を行わないようにするか、あるいは、排他制御を行う必要があります。

Nmc_xxx の関数を1回コールして処理が完結する場合は、マルチスレッド環境でも問題なく動作します。
Nmc_xxx の各関数同士は排他制御を行っています。

また、マルチスレッドと割り込みを併用するアプリケーションを開発する場合は、割り込み発生のタイミングとスレッドからのボードへのアクセスタイミングが重ならないようにしてください。

5.2 プログラミング上の注意点

(1) 入力信号フィルタの初期設定

ボードのリミット信号などの各入力信号は、MCX514内蔵の積分フィルタを使用します。ノヴァエレクトロニクスより供給されるデバイスドライバでは、パソコン起動時の初期設定において、MCX514にを発行して各入力信号に対して、以下のようにフィルタを設定しています。

フィルタ遅延時間：512 μ sec

各信号のフィルタの有効/無効：

信号名	有効 / 無効
EMG	有効
nLMTP, nLMTM	有効
nSTOP0, nSTOP1	有効
nINPOS, nALARM	有効
nPIO6	有効
nSTOP2	有効
nECA, nECB	無効

アプリケーション上で、これらの入力信号フィルタの有効/無効を変更する場合には、MCX514の取扱説明書7.3.6節を参照してください。拡張モード設定コマンド (25h) によって変更することが出来ます。次の記述例では、ボード番号0の全ICのX, Y, Z, U全軸に対して、上記の初期設定と同じ内容を設定しています。Nmc_FilterModeは拡張モード設定コマンド (25h) を実行しています。

(例1)

```
Nmc_FilterMode(0, 0, AXIS_ALL, 0xAA7F); // IC-Aの設定
```

(例2)

```
// IC-Aの設定  
Nmc_WriteReg6(0, 0, 0xAA7F);  
Nmc_WriteReg0(0, 0, 0x0F25);
```

注意：

①コマンドリセット(00FF)を実行した場合にも、各入力信号のフィルタは上記のパソコン起動時初期設定と同じ設定がデバイスドライバ内で行なわれます。

(2) パソコンのスタンバイ、休止動作

本デバイスドライバは、パソコンのスタンバイ動作、あるいは休止動作を行った後のドライバの動作を保証していません。パソコンのスタンバイ動作、あるいは休止動作を行った後に、ボードにアクセスしたい場合は、一度パソコンを再起動させてから行って下さい。

(3) 割り込みサポートについて

割り込みは、VC++およびC#で開発したアプリケーションでサポートしています。本デバイスドライバでサポートしている割り込みの種類は下記の通りです。

- ・RR1レジスタで報告される割り込み全て
- ・連続補間ドライブ、ビットパターン補間でプリバッファのスタックカウンタが8→7、4→3に変わった時の割り込み

(4) 割り込みクリアについて

①RR1レジスタで報告される割り込みについて

ボードでこの割り込みが発生した直後、ドライバ内でRR1を読み出し、割り込みをクリアしています。その後、アプリケーションの割り込み用ユーザー関数が呼び出されます。(ユーザー関数を設定した場合のみ)

②連続補間ドライブ、ビットパターン補間でプリバッファのスタックカウンタが8→7、4→3に変わった時の割り込み

ボードでこの割り込みが発生した直後、ドライバ内で補間割り込みをクリアしています。その後、アプリケーションの割り込み用ユーザー関数が呼び出されます。(ユーザー関数を設定した場合のみ)

5.3 MCX514評価ツール

MCX514 評価ツールプログラムは、MCX514を搭載しているボード(MC8541P/MC8541Pe, MC8581P/MC8581Pe)を評価するツールです。弊社ホームページよりダウンロードすることができます。(MC8000Pデバイスドライバソフトに添付) 評価ツールを実行する前に、MC8000Pデバイスドライバをインストールして下さい。

注：この章では、MCX514 評価ツールの概要について説明します。詳細については、Tool¥MCX514 Boardフォルダの ReadMe.txt(操作説明書)を参照して下さい。

5.3.1 実行プログラムについて

実行プログラムはMCX514.exeです。

32bitOS用はTool¥MCX514 Board¥32 Toolフォルダに、64bitOS用はTool¥MCX514 Board¥64 Toolフォルダにあります。

■評価するMCX514について

各評価ツールが評価するMCX514は下記の通りです。

●MCX514.exe

- ①MCX514を1つ搭載しているボード(MC8541P, MC8541Pe)のMCX514
- ②MCX514を2つ搭載しているボード(MC8581P, MC8581Pe)のMCX514

■実行について

異なるボードに対して複数のMCX514.exeを同時に実行する事ができます。exeの名前を異なる名前に変更してから実行して下さい。(例：MCX514-A.exe、MCX514-B.exe など)

同じボードのICに対して、複数のMCX514.exeを同時に実行することはできません。

■起動時にエラーになる場合

起動時にエラーが発生して実行できない場合は、「3.2.2 サンプルプログラムおよび評価ツールの実行時にエラーが発生する場合の対応」を参照して対応を行ってください。

5.3.2 機能概要

評価ツールを起動すると、ボード番号選択画面が表示され、ボード番号(ボードのロータリースイッチ番号(0~F))を選択することができます。ボード名表示ボタンを押すと、ボード番号の横にボード名(本ドライバを使用しているボードのみ)を表示することができます。ボード番号を選択後、基本動作試験ボタンを押すと、基本動作試験画面が表示されます。

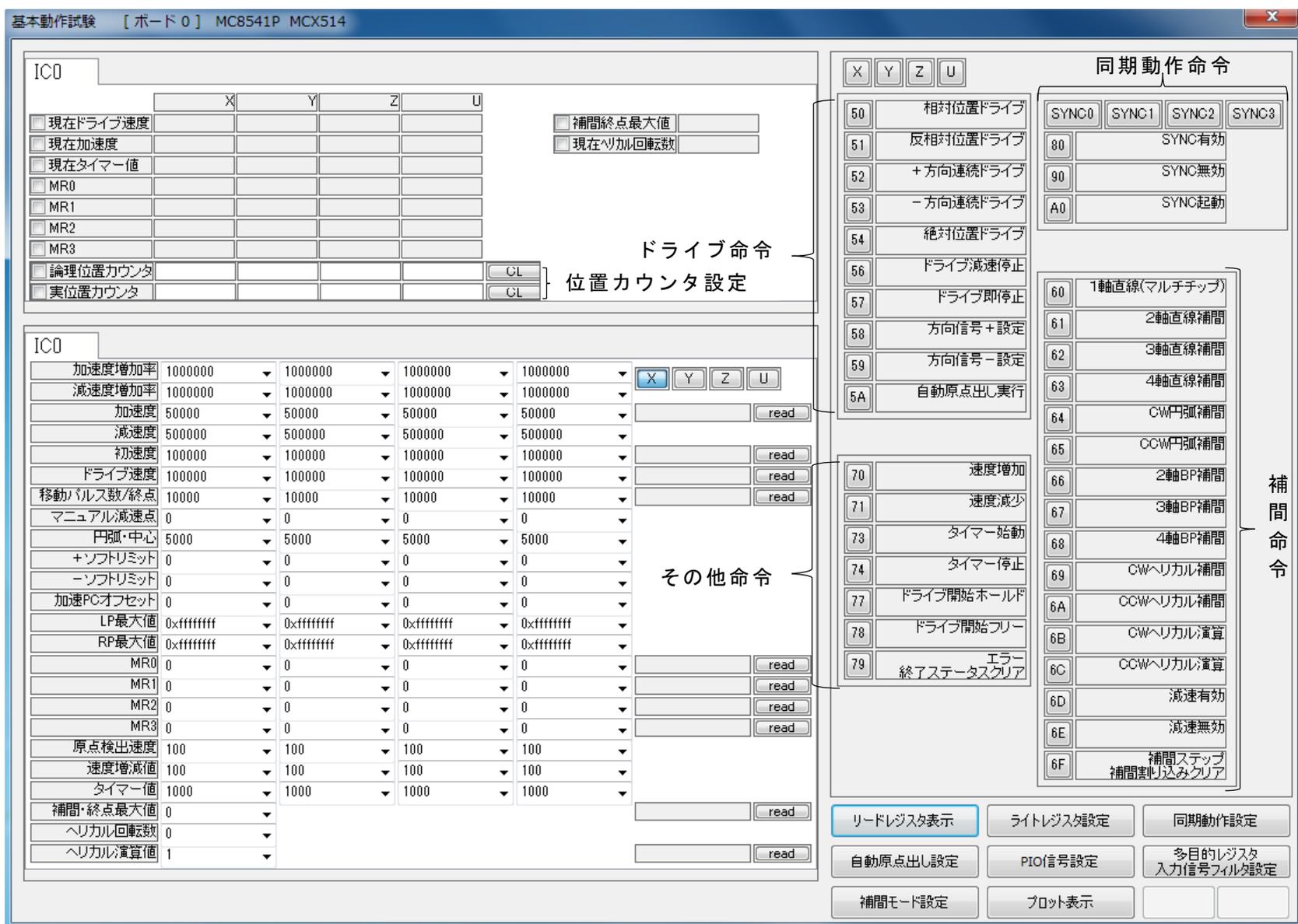
メイン画面では、各軸パラメータ設定および読み出し、ドライブ命令等の命令実行、現在位置・現在速度の表示、割り込み画面表示等を行います。また、モード設定画面(ライトレジスタ・同期動作・自動原点出し・P I O・多目的レジスタ・入力信号フィルタ・補間モード設定画面)やリードレジスタ画面を開き、モード設定、リードレジスタ参照等を行う事ができます。

5.3.3 メイン画面

メイン画面では、下図のような操作を行います。

ドライブ中の現在位置や
現在ドライブ速度等の表示

各軸のドライブ命令等の命令実行

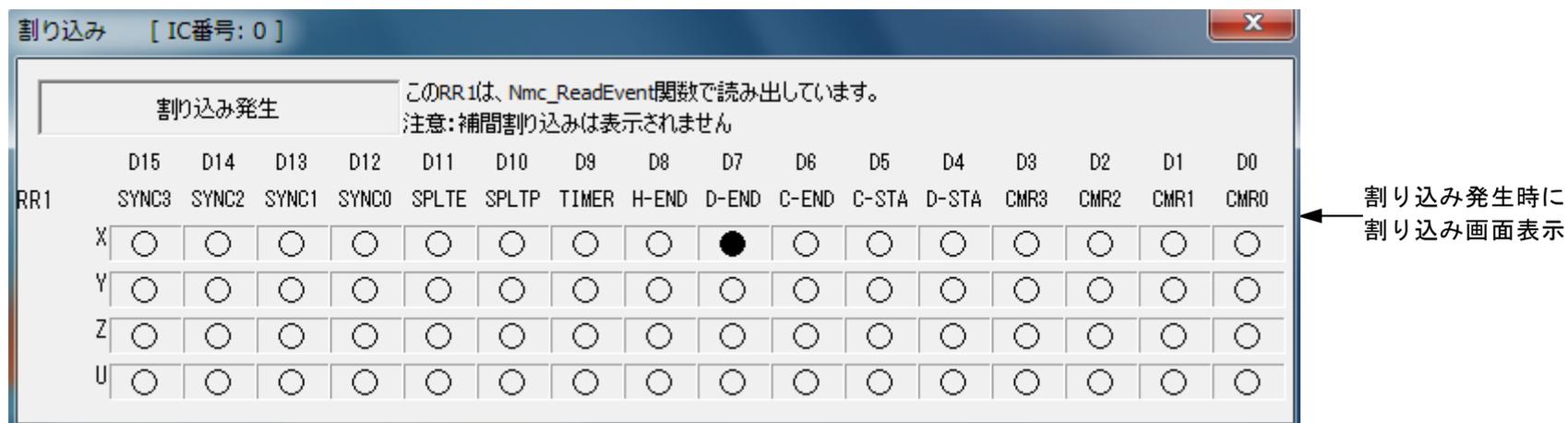


各軸パラメータ設定

各軸パラメータ読み出し

設定画面(6個)
リードレジスタ表示画面の起動

ドライブ中の現在位置や現在ドライブ速度などの表示は、項目の左隣の口をチェックを入れると周期的に値を読み出して表示します。補間ドライブ中の現在ドライブ速度は、主軸の演算パルス速度を読み出して表示します。



割り込み発生時に
割り込み画面表示

5.3.4 ライトレジスタ設定画面

ライトレジスタ設定画面では、MCX514のWR1～WR5（モードレジスタ、アウトプットレジスタ）を設定、WR1～WR3（モードレジスタ）の読み出しをします。

IC0	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
WR1	SYNC3	SYNC2	SYNC1	SYNC0	SPLTE	SPLTP	TIMER	H-END	D-END	C-END	C-STA	D-STA	CMR3	CMR2	CMR1	CMR0
X	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>				
Y	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>				
Z	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>				
U	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>				
WR2	SLM-M	SLM-O	SLM-E	HLM-M	HLM-E	HLM-L	ALM-E	ALM-L	INP-E	INP-L	SP2-E	SP2-L	SP1-E	SP1-L	SP0-E	SP0-L
X	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>										
Y	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>										
Z	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>										
U	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>										
WR3	TMMO	AVTRI	LMINV	PIINV	PI-L	PIMD1	PIMD0	DPINV	DIR-L	DP-L	DPMD1	DPMD0	SACC	DSND	MANLD	
X	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>				
Y	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>				
Z	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>				
U	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>				
WR4	YPI07	YPI06	YPI05	YPI04	YPI03	YPI02	YPI01	YPI00	XPI07	XPI06	XPI05	XPI04	XPI03	XPI02	XPI01	XPI00
	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>				
WR5	UPI07	UPI06	UPI05	UPI04	UPI03	UPI02	UPI01	UPI00	ZPI07	ZPI06	ZPI05	ZPI04	ZPI03	ZPI02	ZPI01	ZPI00
	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>				

WR4およびWR5は全てのボタンがチェックできるようになっていますが、MC8541P/MC8541Pe、MC8581P/MC8581Peでは使用できるPIO信号が限定されています。ハードウェア取扱説明書を参考に正しく設定を行ってください。

5.3.5 同期動作設定画面

同期動作設定画面では、MCX514の同期動作モードレジスタ（SYNC0～SYNC3）を設定します。

The screenshot shows the '同期動作設定画面' (Synchronization Mode Setting Screen) for MCX514. It features four sections for SYNC0, SYNC1, SYNC2, and SYNC3. Each section has a header row with bit positions D15 to D0 and their corresponding functions: REP, AXIS3, AXIS2, AXIS1, SNC+3, SNC+2, SNC+1, ACT4, ACT3, ACT2, ACT1, ACT0, PRV3, PRV2, PRV1, and PRV0. Below each header, there are four rows for axes X, Y, Z, and U. Each axis row contains 16 circular checkboxes, one for each bit position, allowing the user to select which functions are active for that axis and register.

MC8541P/MC8541Pe、MC8581P/MC8581Peでは使用できる同期動作の種類が限定されています。
ハードウェア取扱説明書を参考に正しく設定を行ってください。

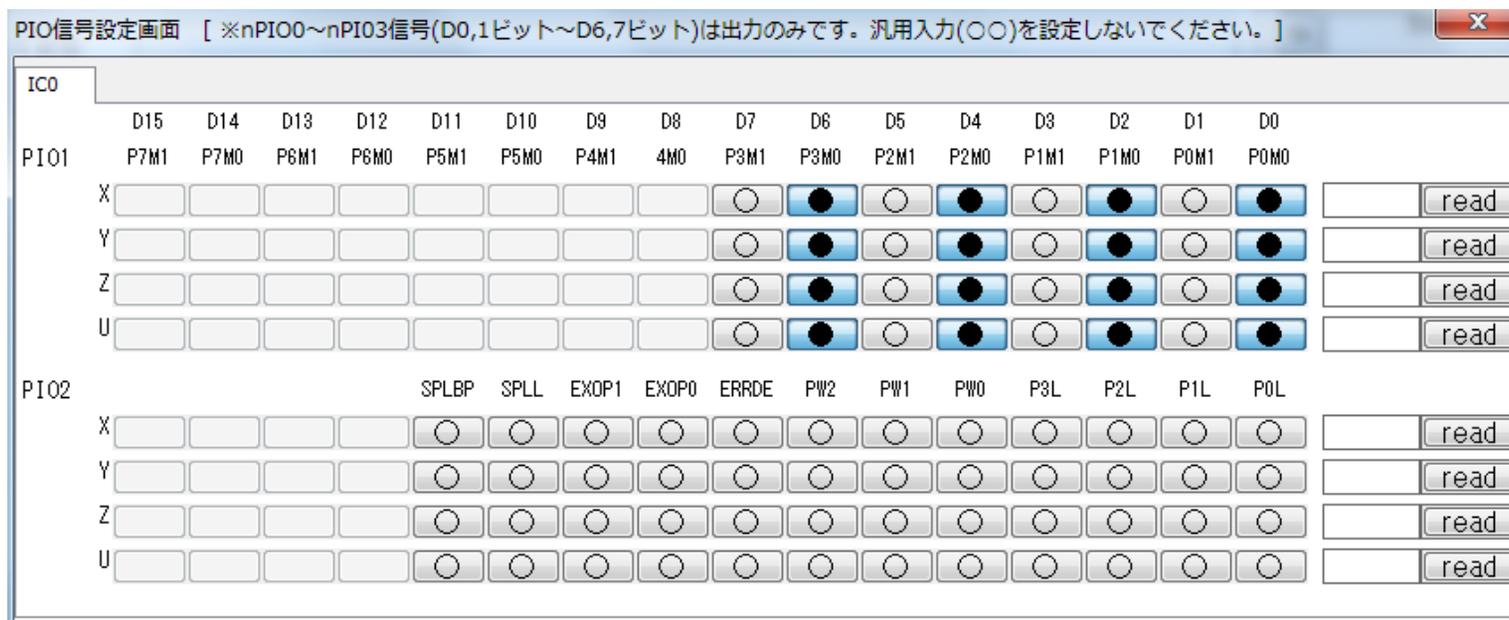
5.3.6 自動原点出し設定画面

自動原点出し設定画面では、MCX514の自動原点出しレジスタ（H1M, H2M）を設定します。

The screenshot shows the '自動原点出し設定画面' (Automatic Home Setting Screen) for MCX514. It features two sections: H1M and H2M. The H1M section has a header row with bit positions D15 to D0 and functions: S4EN, S3LC, S3RC, S3DC, S3DR, S3EN, S2LC, S2RC, S2DC, S2SG, S2DR, S2EN, S1G1, S1G0, S1DR, and S1EN. Below this are four rows for axes X, Y, Z, and U, each with 16 checkboxes. The H2M section has a header row with bit positions D11 to D0 and functions: HTM2, HTM1, HTM0, HTME, DCP2, DCP1, DCP0, DCPL, LCLR, RCLR, and SAND. Below this are four rows for axes X, Y, Z, and U, each with 10 checkboxes.

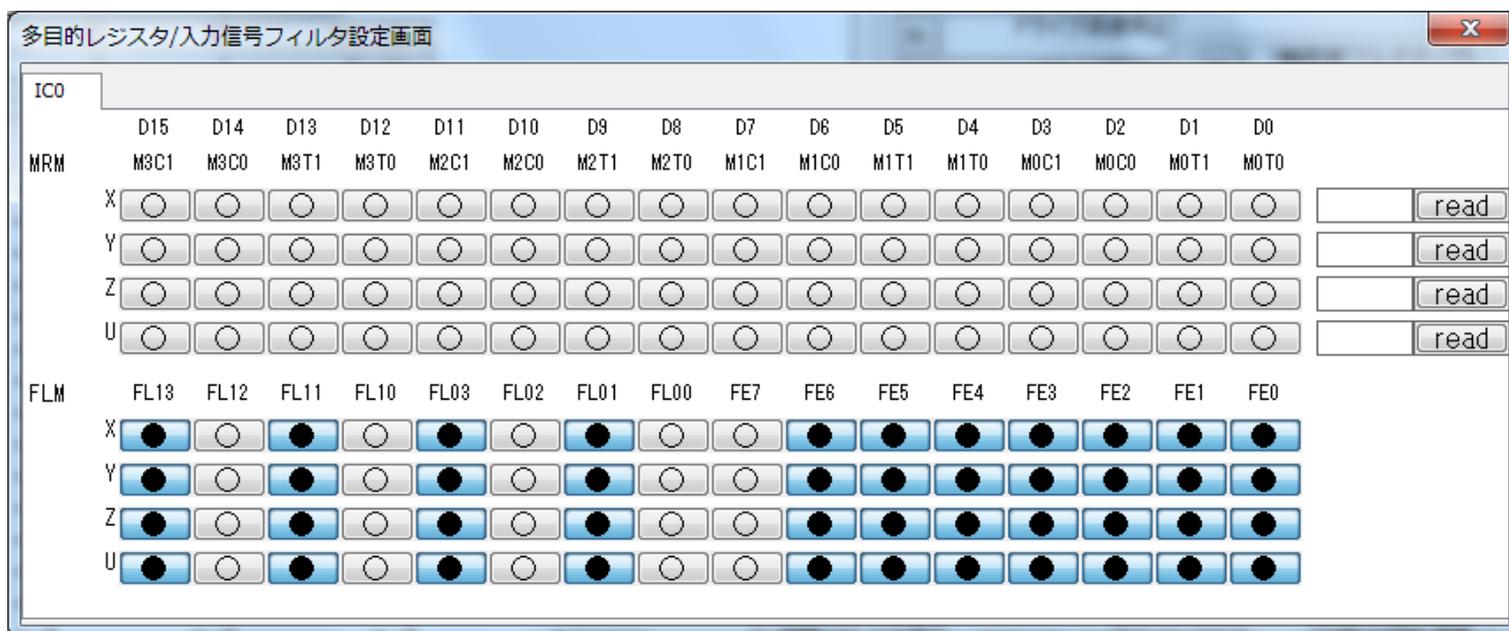
5.3.7 PIO信号設定画面

PIO信号設定画面では、MCX514のPIO (PI01,PI02) を設定、読み出しをします。



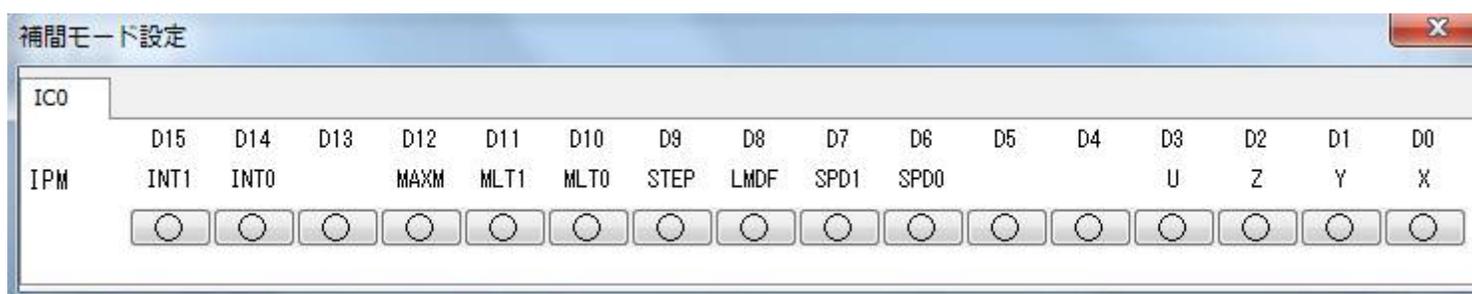
5.3.8 多目的レジスタ/入力信号フィルタ設定画面

多目的レジスタ/入力信号フィルタ設定画面では、MCX514の多目的レジスタ (MRM) の設定および読み出し、入力信号フィルタ設定 (FLM) の設定をします。



5.3.9 補間モード設定画面

補間モード設定画面では、MCX514の補間モード (IPM) を設定します。



補間ドライブの評価が終了し、他の評価をおこなう際には、必ず全設定ボタンをクリアしてください。

5.3.10 リードレジスタ画面

リードレジスタ画面では、RR0, RR2, RR3, RR4, RR5（ステータスレジスタ、P I Oリードレジスタ）の現在の値を一定間隔でMCX514から読み出し、画面に表示します。データの読み出し間隔は、50ミリ秒間隔です。
RR1 については、割り込み発生時、割り込み画面に表示します。

リードレジスタ [※RR1は割り込み発生時、割り込み画面に表示します]

IC0	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
RR0	HSTCK3	HSTCK2	HSTCK1	HSTCK0	CNEXT	ZONE2	ZONE1	ZONE0	U-ERR	Z-ERR	Y-ERR	X-ERR	U-DRV	Z-DRV	Y-DRV	X-DRV
	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
RR2	EMG	ALARM	LMT-	LMT+	STOP2	STOP1	STOP0	SYNC	CERR	HOME	EMG	ALARM	HLMT-	HLMT+	SLMT-	SLMT+
X	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Y	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Z	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
U	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
RR3	PAGE	HSST5	HSST4	HSST3	HSST2	HSST1	HSST0	LMTM	LMTP	ALARM	INPOS	ECB	ECA	STOP2	STOP1	STOP0
X	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>								
	PAGE	HSST5	HSST4	HSST3	HSST2	HSST1	HSST0	LMTM	LMTP	ALARM	INPOS	ECB	ECA	STOP2	STOP1	STOP0
Y	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>								
	PAGE	HSST5	HSST4	HSST3	HSST2	HSST1	HSST0	LMTM	LMTP	ALARM	INPOS	ECB	ECA	STOP2	STOP1	STOP0
Z	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>								
	PAGE	HSST5	HSST4	HSST3	HSST2	HSST1	HSST0	LMTM	LMTP	ALARM	INPOS	ECB	ECA	STOP2	STOP1	STOP0
U	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>								
RR4	YPI07	YPI06	YPI05	YPI04	YPI03	YPI02	YPI01	YPI00	XPI07	XPI06	XPI05	XPI04	XPI03	XPI02	XPI01	XPI00
	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
RR5	UPI07	UPI06	UPI05	UPI04	UPI03	UPI02	UPI01	UPI00	ZPI07	ZPI06	ZPI05	ZPI04	ZPI03	ZPI02	ZPI01	ZPI00
	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

