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

改訂　　２０２４．　４．１２　[Ver 9.0.0.0]

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
	MC8543PeL

NOVA electronics　　株式会社 ノヴァエレクトロニクス

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: https://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

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ＩＣ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
	MC8543PeL

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
Windows10(64bit) Windows11

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
Microsoft 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
MC8543PeL	MCX514	1	4

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

ホームページからデバイスドライバをダウンロードし、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、異なる機種の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。

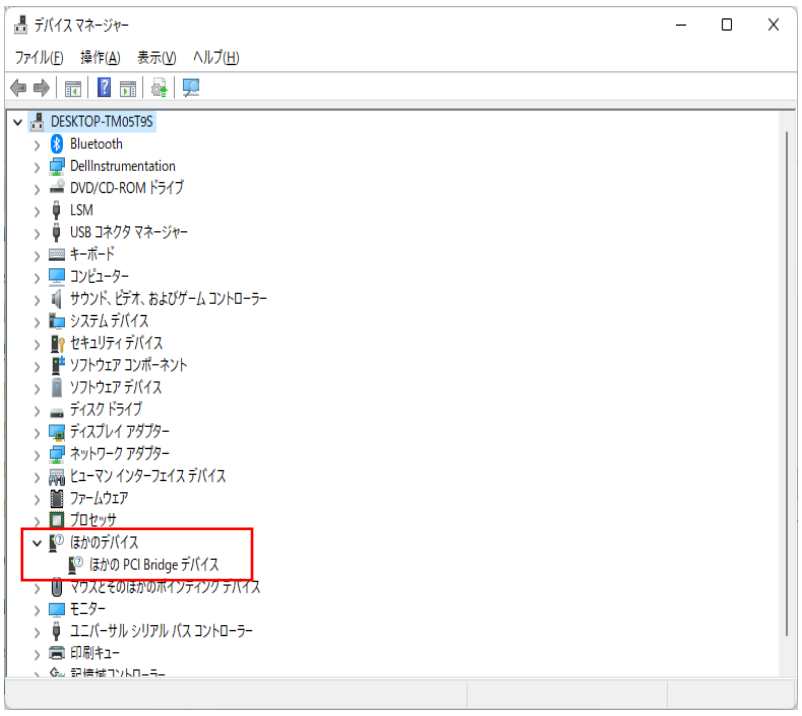
① パソコン本体の電源がＯＦＦであることを確認してから、外装カバー､スロットカバー等を外します。
② 空いている拡張スロットへ本ボードを差し込みます｡基板のエッジコネクタをパソコンのPCIバス/ PCI Expressコネクタに正しく挿入してください。
③ 取付金具をネジ止めしてください。この時キチンとねじを締めないと後で抜け落ちたりするなどして､ショートや故障､誤動作の原因となります。
④ パソコン本体の外装カバーを元通りに取り付けます。

2.4　デバイスドライバのインストール

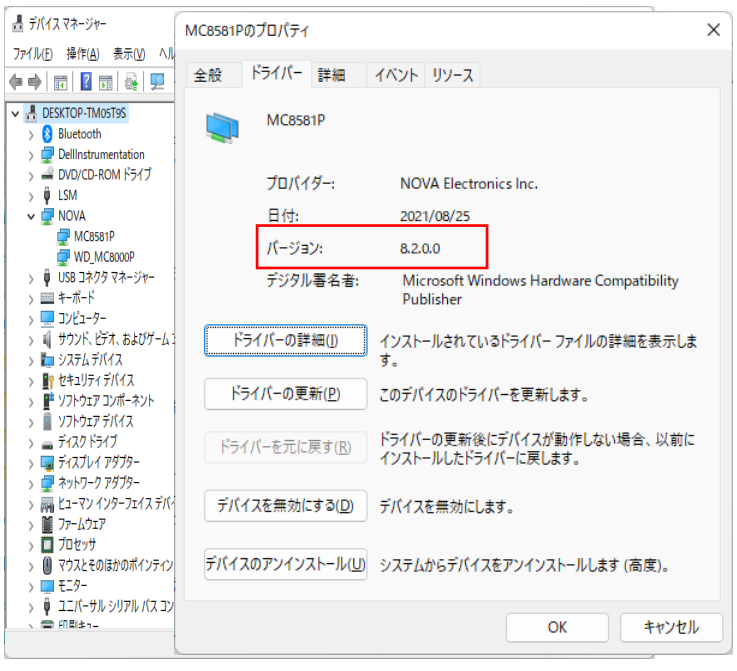
この章では、本ボードのパソコンへの取り付けとデバイスドライバのインストール方法について説明します。

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管理者アカウントでログインしてから行ってください。
下記の手順はWindows 11を例にしていますが、Windows 10(64bit)でもインストールの進め方は同じです。

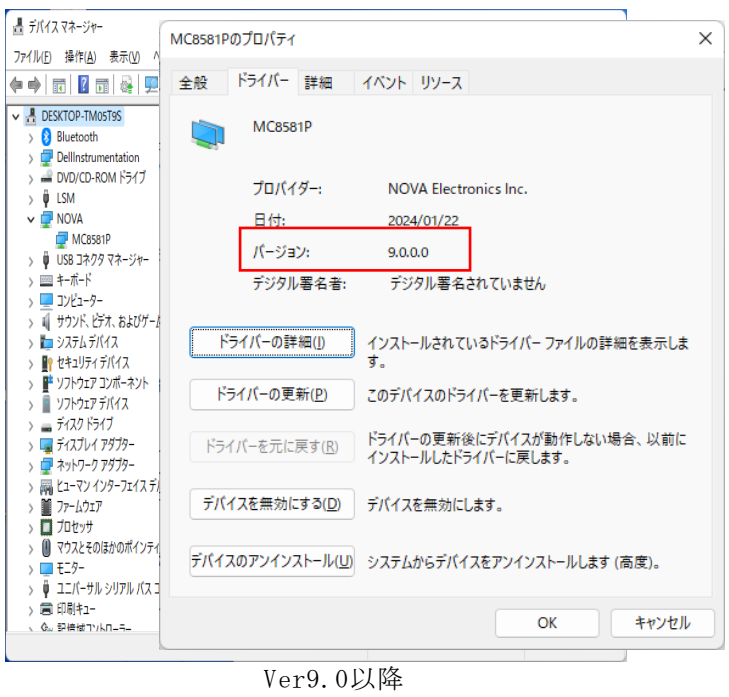
① 2.1の方法でインストールするデバイスドライバを準備して下さい。
② 2.2, 2.3 を実行し、本ボードが確実にパソコンに取り付けられているか確認してください。
③ パソコン本体の電源をＯＮし、Windows 11を起動してください。
④ 管理者アカウントでログインしてください。
⑤ インストールするパソコンのMC8000Pデバイスドライバのインストール状況を確認してください。
デバイスマネージャーを確認し、下記画面のよう「ほかのPCI Bridgeデバイス」となっていることを確認します(MC8000Pデバイスドライバがインストールされていないことを確認します)。



MC8000Pデバイスドライバがインストールされている場合、先にアンインストールを行います。
プロパティ画面からバージョンを確認します。Ver8.XX(XXは数字が入ります)の場合(下記画面の左)は2.5.3項を、Ver9.0以降の場合(下記画面の右)は2.5節を参照してアンインストールしてください。

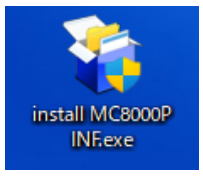


Ver8.XX

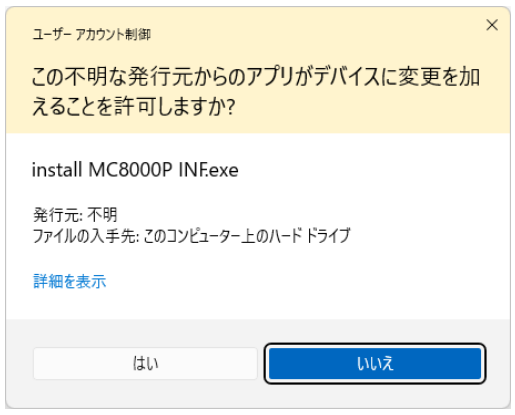


Ver9.0以降

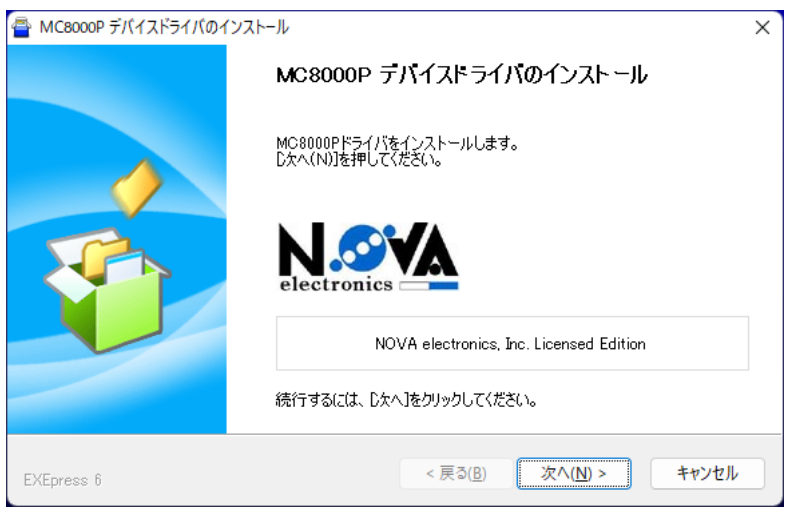
⑥　ダウンロードしたソフトウェアのDriverフォルダを選択し、「install MC8000P INF.exe」をダブルクリックしてください。



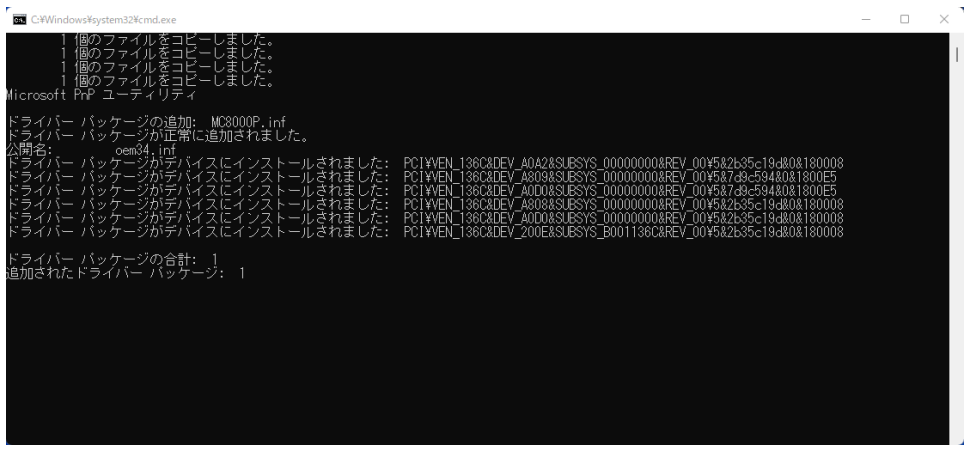
⑦　「ユーザーアカウント制御」の画面が出てきた場合は、[はい]ボタンをクリックして操作を進めてください。



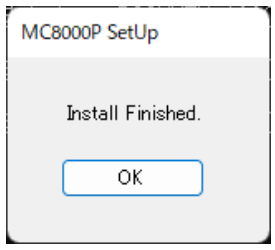
⑧　デバイスドライバのインストール画面
［次へ］のボタンをクリックしてください。



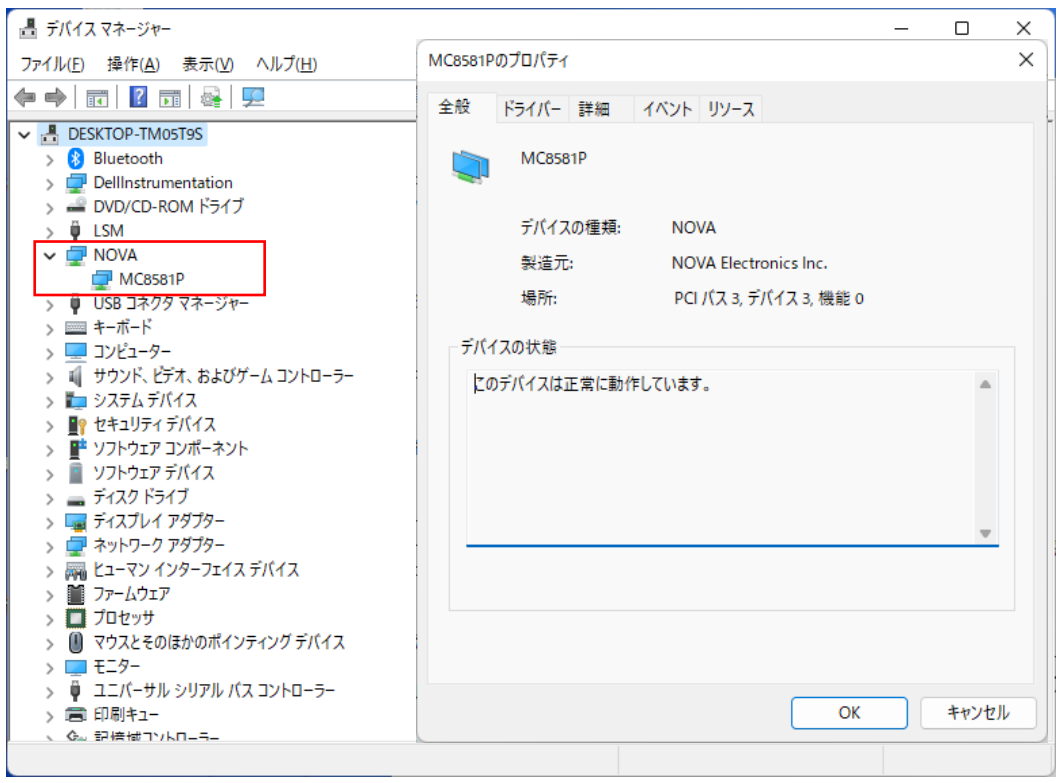
⑨　次のような画面が表示され、処理が終了すると自動的に画面が閉じます。（インストールに時間がかかることがあります。）



⑩　［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。



⑪　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。
下記左画面のデバイスマネージャーを開き、[NOVA]の下にボード名[XXXX]が表示されていることを確認します。
（XXXX にはお使いのボード名が入ります。下記画面例は MC8581P となります。）



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」と表示されていたらインストールは正常終了です。
(画面上のMC8581P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。

2.4.1　異なる種類のボードを複数枚使用する場合のインストール

ご使用のパソコンにデバイスドライバがインストールされていない場合は、2.4節の手順でインストールしてください。
Ver9.0以降のデバイスドライバがインストールされている場合は、インストールは不要となります。2.2節、2.3節を実行してパソコンの電源をON後、デバイスマネージャーを確認してインストールされていることを確認してください。
Ver8.XX(XXは数字が入ります)のデバイスドライバがインストールされている場合は、2.5.3項の手順でアンインストールを行った後、2.4節の手順でインストールしてください。

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 11を例にしていますが、Windows 10(64bit)でもアンインストールの進め方は同じです。

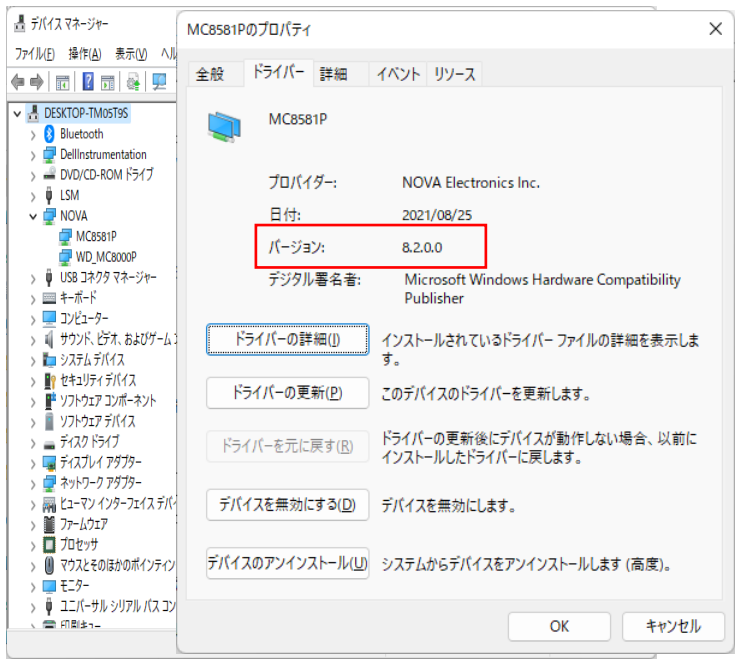
2.5.1　デバイスドライバのアンインストール

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（MC8581PとMC8082P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「2.5.2　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」を参照して作業を行ってください。

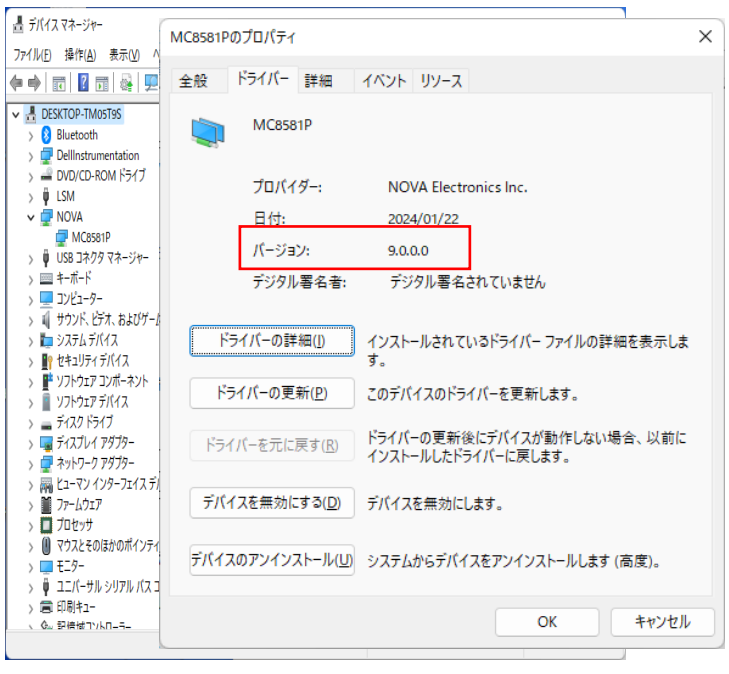
2.5.1.1　Uninstall MC8000P INF.exeの実行

①　管理者アカウントでログインしてください。
②　デバイスマネージャーを確認し、MC8000Pデバイスドライバのバージョンを確認してください。

Ver8.XX(XXは数字が入ります)の場合は、2.5.3項を参照してアンインストールしてください。

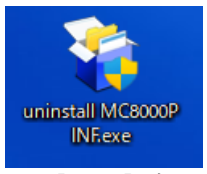


Ver9.0以降のバージョンの場合、ご使用になるソフトウェアのバージョンを確認してください。

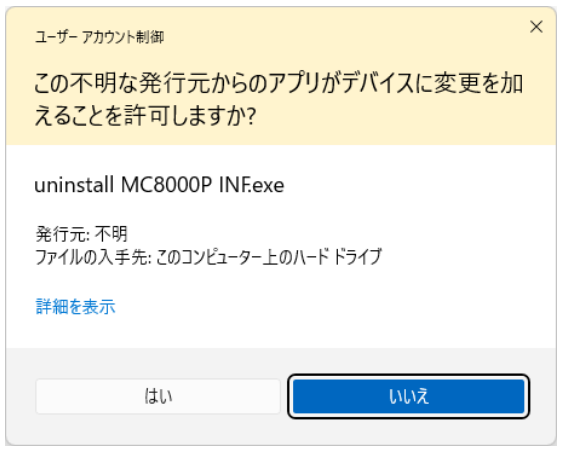


③　ダウンロードしたソフトウェアのデバイスドライバのバージョンを確認してください。
②で確認したバージョンと異なる場合、該当するバージョンのソフトウェアをダウンロードしてください。

④　ダウンロードしたソフトウェアのDriverフォルダを選択し、「uninstall MC8000P INF.exe」をダブルクリックしてください。



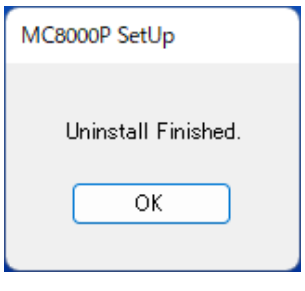
⑤　「ユーザーアカウント制御」 の画面が表示されたら、[はい]ボタンをクリックして操作を進めてください。



⑥　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　してください。

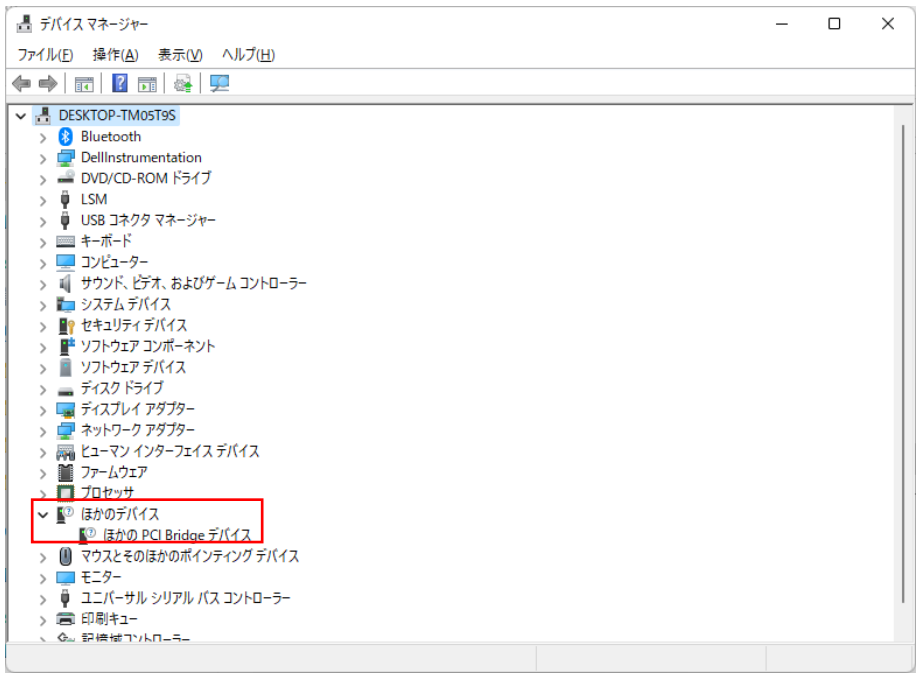
⑦　処理が終了すると自動的に画面が閉じます。（アンインストールに時間がかかることがあります。）

⑧　［ＯＫ］ボタンをクリックしてください。

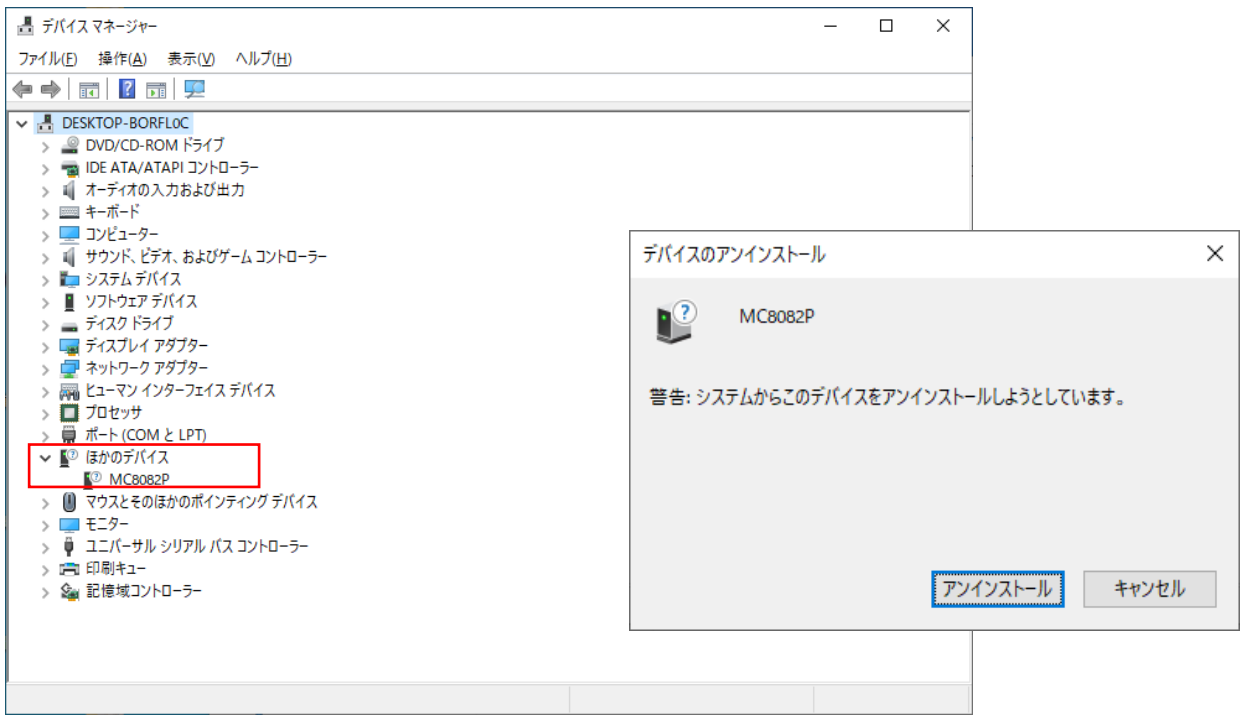


2.5.1.2　デバイスマネージャーの確認

①　デバイスマネージャーで本ボードが削除されていることを確認して下さい。



下記画面のように、完全に削除されなかった場合、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[デバイスのアンインストール]を選択後、[アンインストール]のボタンをクリックしてドライバを削除してください。



②　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てください。
③　ボードを止めているビスを外してください。
④　本ボードを指先でつまんで軽く左右にゆするようにしながら引き出し、PCから外してください。

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（MC8581PとMC8082P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。

MC8581PとMC8082Pの2種類のボード2枚が存在する場合を例にとり説明します。
2枚とも取り外す場合は、2.5.1項の手順でアンインストールをおこなってください。
どちらか1枚だけ取り外す場合、例えばMC8082Pだけ取り外す場合(MC8581Pは使用する場合)、アンインストールは不要です。

2.5.3　Ver8.XX(XXは数字が入ります)のデバイスドライバのアンインストール

Ver8.XX(XXは数字が入ります)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。

Ver9.0以降のデバイスドライバ一式にはVer8.XX(XXは数字が入ります)のデバイスドライバのアンインストーラは含まれていません。
アンインストールをおこなうバージョンのデバイスドライバを用意して下さい。
ＣＤ－ＲＯＭからデバイスドライバをアン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アン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。
ＣＤ－ＲＯＭやダウンロードソフトウェアが手元にない場合、2.5.3.3項を参照してアンインストールを行ってください。

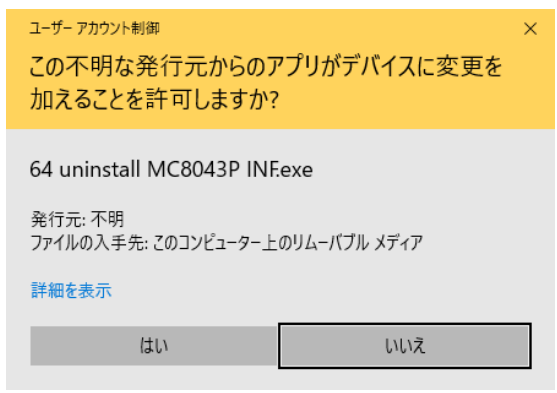
2.5.3.1　Uninstall XXXX INF.exeの実行

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。

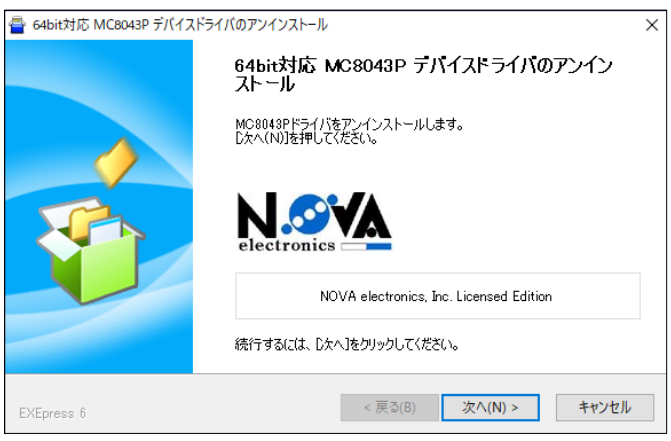
① 管理者アカウントでログインしてください。
② 提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、64 Driverフォルダ内の「64 uninstall XXXX INF.exe」をダブルクリックしてください。
「64 uninstall XXXX INF.exe」はデバイスドライバのバージョンVer8.XX(XXは数字が入ります)に対応したものを使用してください。
（画面上のMC8043Pはお使いのボード名に置き換えてご覧下さい。）

「64 uninstall XXXX INF.exe」が手元になくダウンロードも出来ない場合、2.5.3.3項を参照してアンインストールを行ってください。

③ 「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MC8043Pはお使いのボード名に置き換えてご覧下さい。）

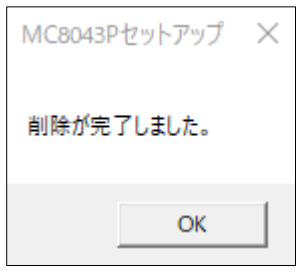


④ デバイスドライバのアンインストール画面
[次へ]のボタンをクリック してください。（画面上のMC8043Pはお使いのボード名に置き換えてご覧下さい。）



⑤　処理が終了すると自動的に画面が閉じます。（アンインストールに時間がかかることがあります。）

⑥　［ＯＫ］ボタンをクリックしてください。

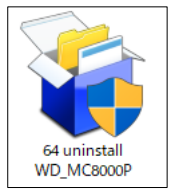


続いて、「2.5.3.2　Uninstall WD_MC8000P.exeの実行」を行います。

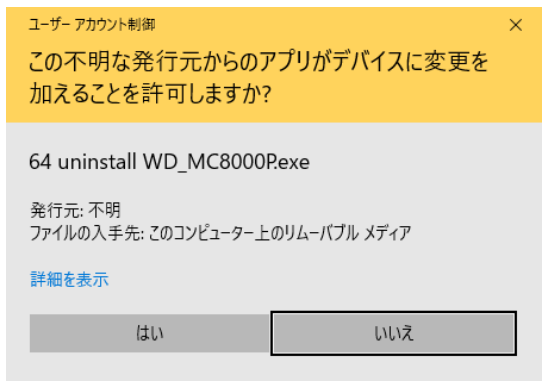
2.5.3.2　Uninstall WD_MC8000P.exeの実行

Uninstall WD_MC8000P.exeを以下の手順で実行します。

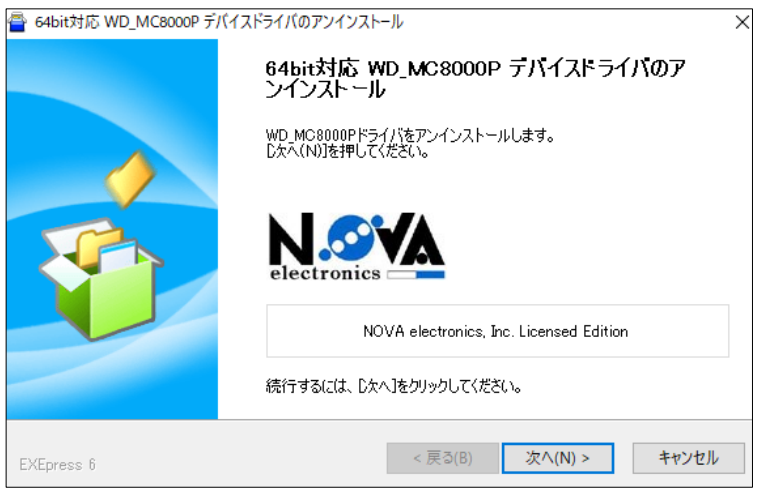
①　管理者アカウントでログインしてください。
②　提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、64 Driverフォルダ内の「64 uninstall WD_MC8000P.exe」をダブルクリックしてください。



③　「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。



④　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してください。



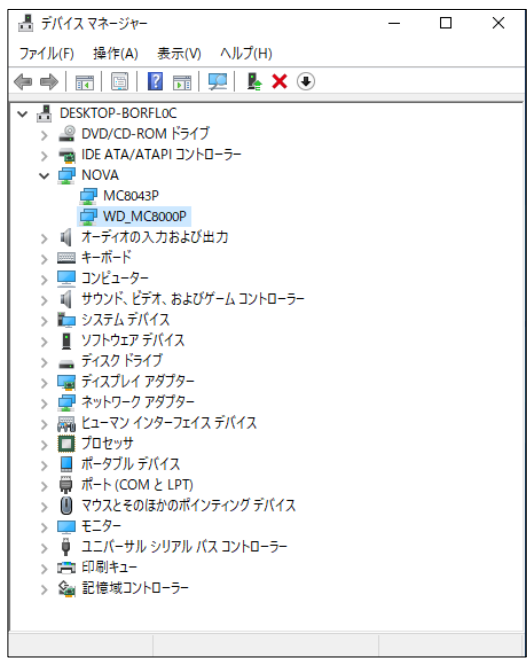
⑤　処理が終了すると自動的に画面が閉じます。（アンインストールに時間がかかることがあります。完了のメッセージがあるまでお待ちください。）

⑥　［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。

2.5.3.2　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]－[ハードウエアとサウンド]－[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。

② 取り外すボードがMC8043Pとした場合、uninstall MC8043P INF.exeとUninstall WD_MC8000P.exeを実行し、MC8043PおよびWD_MC8000Pが削除されていることを確認して下さい。



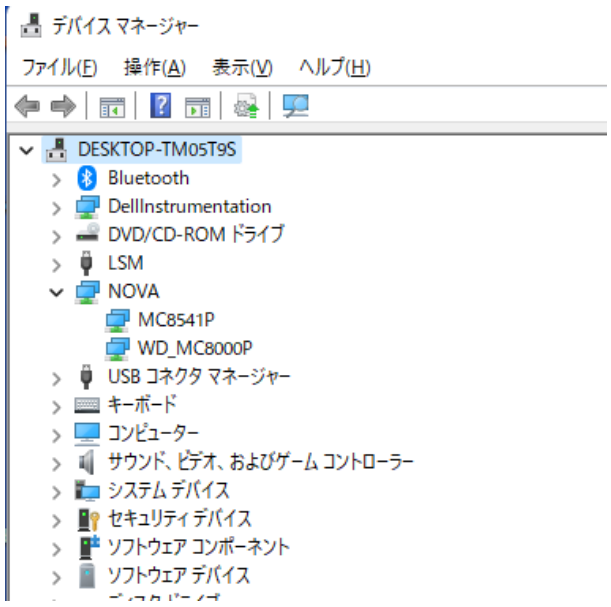
削除されていない場合は、削除されていないドライバをクリックし、[操作]メニューから[デバイスのアンインストール]を選択後、[アンインストール]のボタンをクリックしてドライバを削除してください。

③ 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てください。
④ ボードを止めているビスを外してください。
⑤ 本ボードを指先でつまんで軽く左右にゆするようにしながら引き出し、PCから外してください。

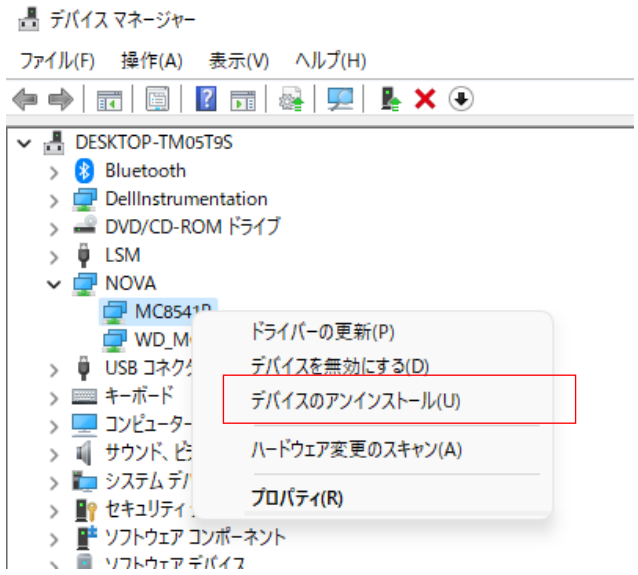
2.5.3.3　Ver8.XX(XXは数字が入ります)のデバイスドライバの手動アンインストール

アンインストールはアンインストール実行ファイル「Uninstall XXXX INF.exe(XXXXにはお使いのボード名が入ります)」で行いますが、ファイルが手元にない場合、以下の手順で手動アンインストールを行います。

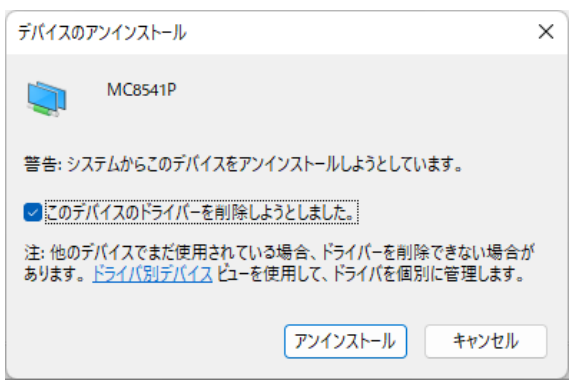
① ［コントロールパネル］－［システムとセキュリティ］－［ハードウエアとサウンド］－［デバイスマネージャー］タブを開きます。



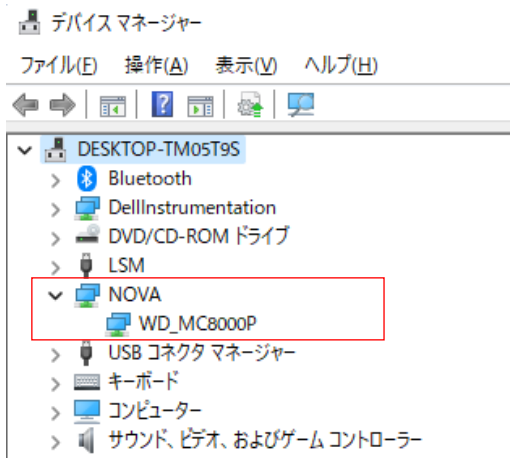
② ボードを選択し(下記画面では MC8541P)、右クリックで表示されるメニューから「デバイスのアンインストール(U)」を実行します。



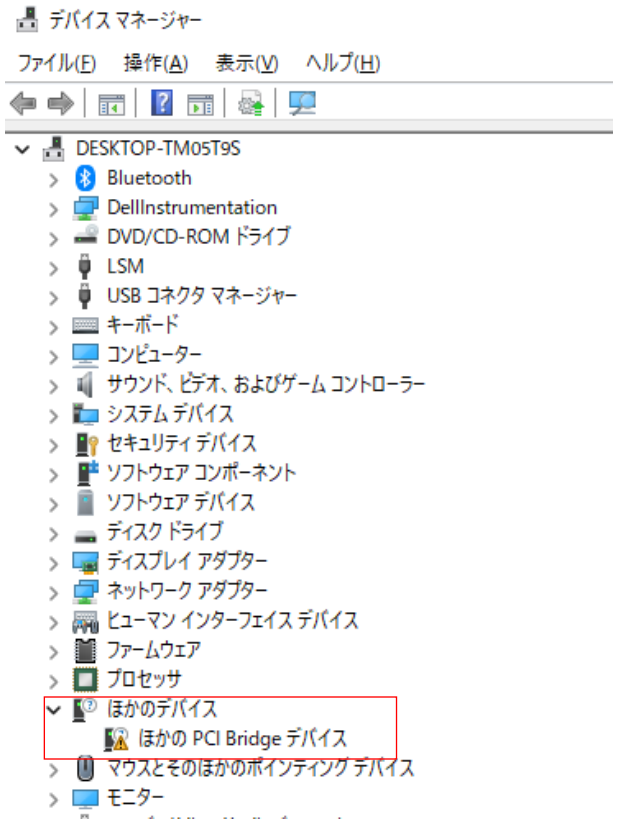
③ 表示されたダイアログの「このデバイスのドライバーを削除しようとしました。」にチェックを入れ、アンインストールを実行します。



④　デバイスマネージャーからボード情報が削除されていることを確認します。



⑤　WD_MC8000P に関して、同様に②、③の手順でアンインストールを行います。
⑥　デバイスマネージャーで WD_MC8000P が削除され、下記画面のように「ほかの PCI Bridge デバイス」となっていることを確認します。

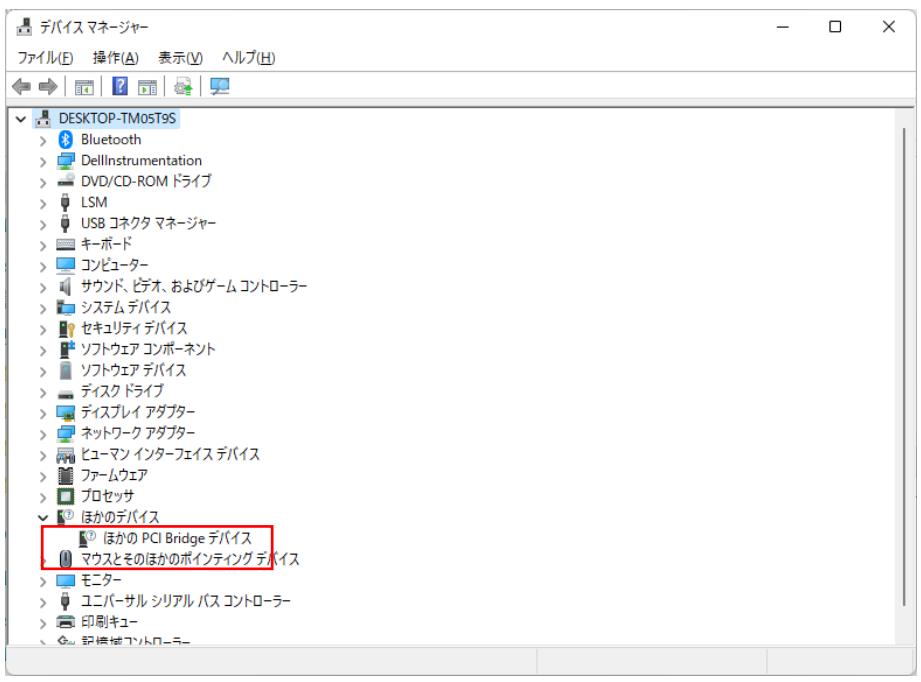


2.6　デバイスドライバの更新

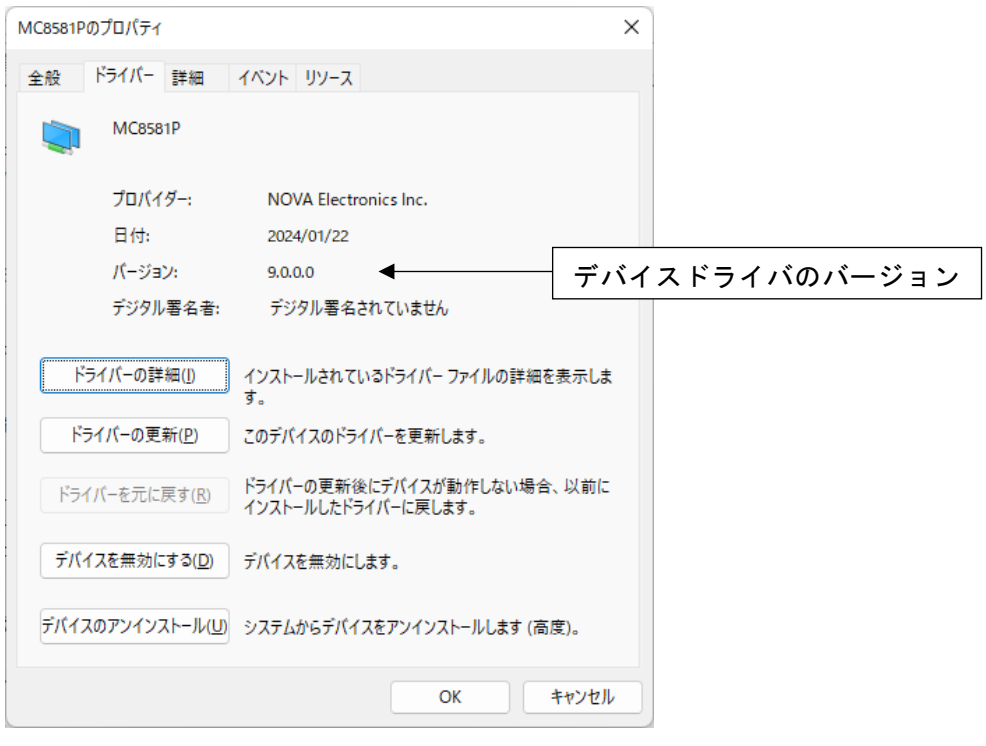
デバイスドライバのバージョンが新しくなった場合、次の手順でドライバの更新を行ってください。
以下に、更新手順を説明します。

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管理者アカウントでログインしてから行ってください。

① デバイスマネージャーを確認し、Ver8.XX(XXは数字が入ります)の場合は2.5.3項を、Ver9.0以降は2.5節を参照してデバイスドライバをアンインストールしてください。
② パソコンを再起動してください。
③ デバイスマネージャーを開き、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。



④ 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。
⑤ 2.4の⑥から⑪を参照してインストールします。
⑥ ダウンロードしたソフトウェアのDriverフォルダのMC8000P_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。
（画面上のMC8581P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 11, Windows 10(64bit)

対応言語

Microsoft Visual C++ 2019　以降
Microsoft Visual Basic 2019　以降
Microsoft Visual C# 2019　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マイクロソフト公式ホーム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 2019以降で動作させる場合は、マイクロソフト公式ホーム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。
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。

64bitPCで32bitアプリケーションを動作させたい場合は、弊社までご連絡ください

3.2 ソフトウェア構成

3.2.1 ソフトウェアー覧

項目	フォルダ	ファイル名	説　　明
デバイスドライバ	Driver	install MC8000P INF.exe	インストールプログラム
		uninstall MC8000P INF.exe	アンインストールプログラム
	Lib¥VC	MC8000P.LIB	MC8000P.DLLを使用するためのライブラリ　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: https://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	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	
		Parameter Sample	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搭載ボード評価ツール ※MC8043P	Tool ¥MCX314As Board	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。
		Parameter Sample	パラメータサンプルファイル： MCX314As-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
MCX514搭載ボード評価ツール ※MC8541P、MC8581P	Tool ¥MCX514 Board(MC8541P_MC8581P)	MCX514.exe	MCX514搭載ボードMC8541P、MC8581Pの評価ツール。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 ※MC8543PeL	Tool ¥MCX514 Board(MC8543PeL)	MCX514_PeL.exe	MCX514搭載ボードMC8543PeLの評価ツール。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。

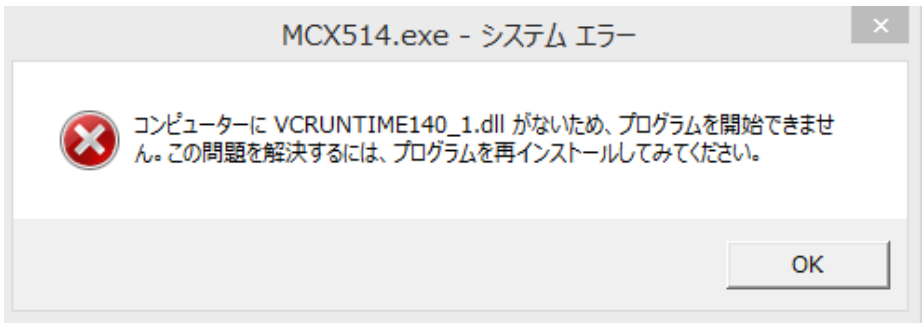
備考：VC++のMFC AppWizerdが自動的に作成するファイルに関しては説明を省略します。
MCX514搭載ボードの評価ツールは、MCX514.exeとMCX514_PeL.exe の二つ存在しますが、ご使用のボードによって使用可能な評価ツールが決まっています。MC8541P、MC8581Pをご使用の場合、MCX514.exeを使用してください。MC8543PeLでは使用出来ません。
同様に、MC8543PeLをご使用の場合、MCX514_PeL.exeを使用してください。MC8541P、MC8581Pでは使用出来ません。

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 2015用、評価ツールはVisual Studio 2019用のものをインストールしてください。
例えば、評価ツールのエラーを解除する場合は、Visual Studio 2019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 2015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。

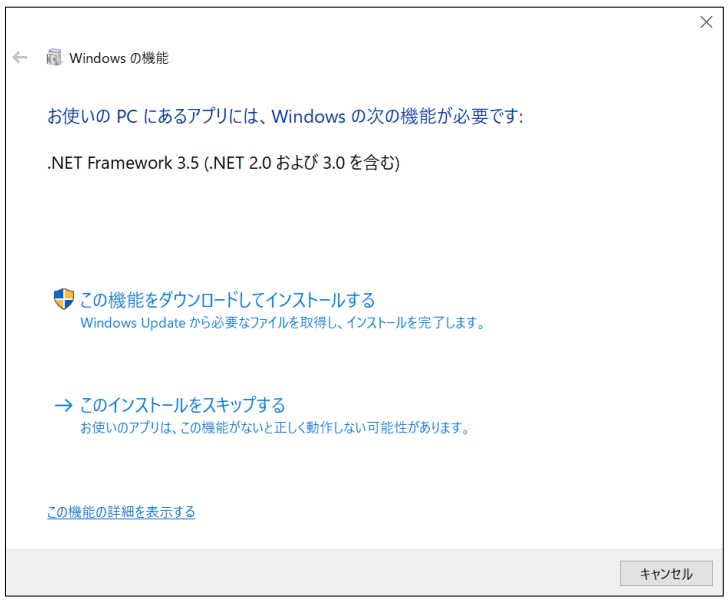


評価ツール実行時のエラー例

■ 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を有効にして下さい。

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++ 2019 以降対応です。

① Lib¥VC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。

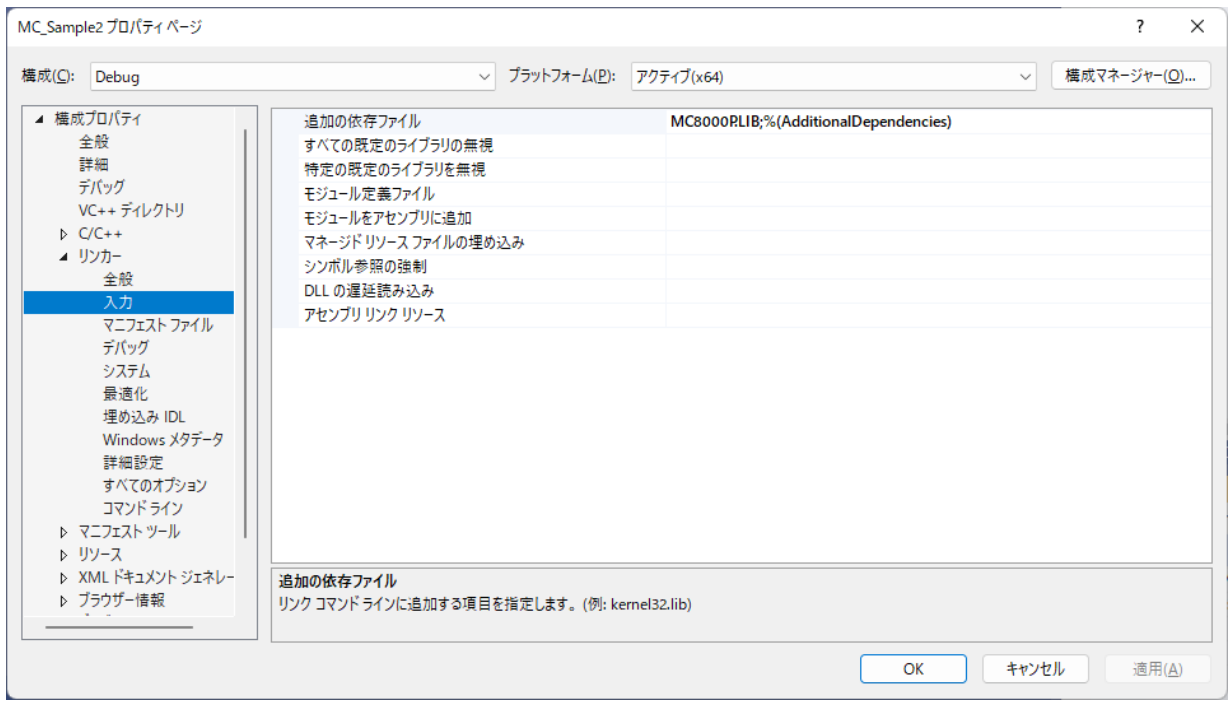


図3.3-1 VC++ 2019 プロジェクトのプロパティ

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 VB2019 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。

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¥C#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　を追加してください。

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。

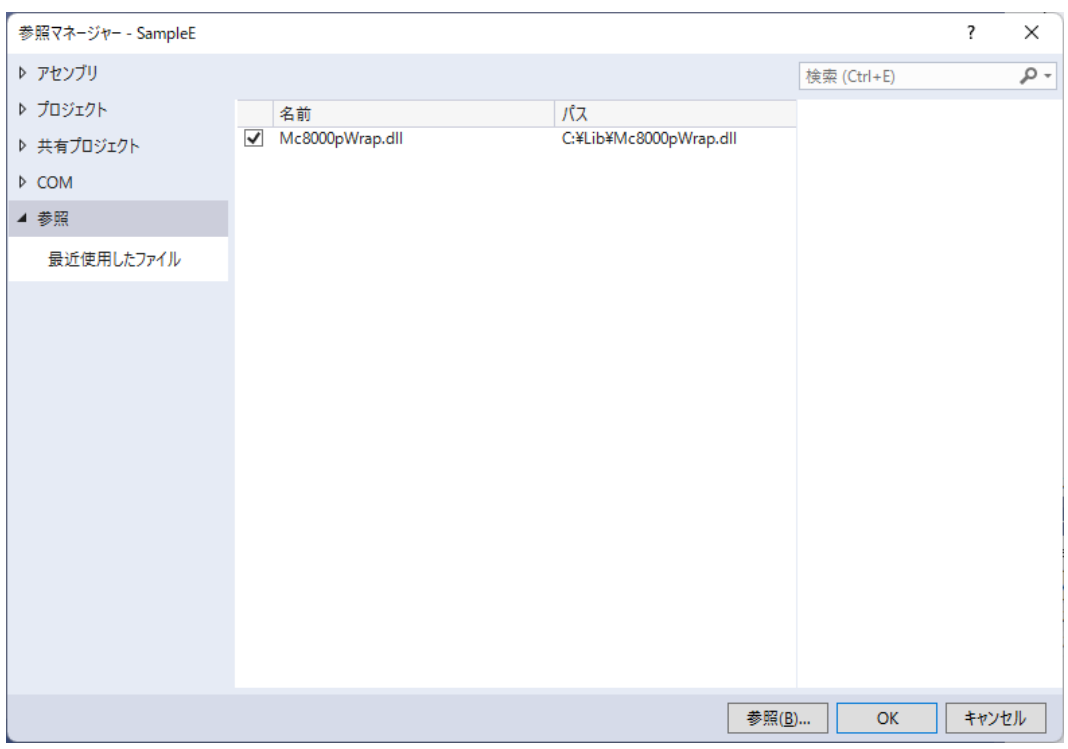


図 3.3-2 C# 2019 参照の追加

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ＩＣ	ＩＣ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が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	○	○	○	23	
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	○	○	○	24	
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	○	○	○	25	
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	○	○	○	26	
Nmc_SetEvent	割込みを処理するユーザー関数を設定する	○	○	○	27	
Nmc_ResetEvent	割込みを処理するユーザー関数の設定を解除する	○	○	○	28	
Nmc_ReadEvent	割込み発生直後の各軸ＲＲ３の値を取得する	○	○	○	28	

（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	○	○	○	29	
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	○	○	○	30	
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	○	○	○	31	
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	○	○	○	32	
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	○	○	○	32	
Nmc_ReadReg1	ＲＲ１(ステータスレジスタ１)　読み出し	○	○	○	33	
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	○	○	○	34	
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	○	○	○	35	
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	○	○	○	36	
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)	○	×	○	37	
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)	○	×	○	38	
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	○	○	○	39	
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	○	○	○	40	
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	○	○	○	41	

（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	○	○	○	41	※１
Nmc_SyncMode	同期動作モード設定	○	○	○	42	※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	○	○	○	43	
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	○	○	○	44	
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	○	○	○	45	
Nmc_GetCNextStatus	連続補間次データ書込み可能状態取得	○	○	○	〃	※１
Nmc_GetBpSc	ＢＰ補間スタックカウンタ取得	○	○	○	46	※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)	○	○	○	46	
Nmc_ReadRegSetAxis	軸指定リードレジスタ読み出し(RR1～2)	○	○	○	47	
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）	○	○	○	48	※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ＢＰ補間実行	○	○	○	49	※１
Nmc_3BPExec	３軸ＢＰ補間実行	○	○	○	51	※１
Nmc_2BPExec_BG	２軸ＢＰ補間実行（バックグラウンドで実行）	○	○	○	53	※１
Nmc_3BPExec_BG	３軸ＢＰ補間実行（バックグラウンドで実行）	○	○	○	56	※１
Nmc_2CIPExec	２軸連続補間実行	○	○	○	59	※１
Nmc_3CIPExec	３軸連続補間実行	○	○	○	61	※１
Nmc_2CIPExec_BG	２軸連続補間実行（バックグラウンドで実行）	○	○	○	63	※１
Nmc_3CIPExec_BG	３軸連続補間実行（バックグラウンドで実行）	○	○	○	66	※１
Nmc_IPStop	補間実行を中断する	○	○	○	69	※１
Nmc_IPGetMsgNo	補間終了時の受信ﾒｯｾｰｼﾞからﾎﾞｰﾄﾞ番号と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	ボードの使用を開始する。 **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); **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。 **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 **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);
Nmc_Close	ボードの使用を終了する。 **VC**　BOOL　Nmc_Close(int No); **VB.NET**　Function Nmc_Close(ByVal No As Integer) As Integer **C#**　bool　MC8000P.Nmc_Close(int No); **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)） **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 **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);
Nmc_CloseAll	全てのボードの使用を終了する。 **VC**　BOOL　Nmc_CloseAll(void); **VB.NET**　Function Nmc_CloseAll() As Integer **C#**　bool　MC8000P.Nmc_CloseAll(); **入力パラメータ** なし **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 **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();

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_GetBoardInfo</td><td>
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。

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);

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.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。

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

使用例
　[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)
</td></tr>
<tr><td>Nmc_OutPort</td><td>
出力ポートに２バイトデータを書き込む。

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);

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「4.1.3 補足説明」(1)①、(6)参照。
　Data　書き込むデータ

戻り値
　なし

使用例
　[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);
</td></tr>
</table>

関数名	機能　及び　内容
Nmc_InPort	入力ポートから２バイトデータを読み出す。 **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); **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。 **戻り値** 入力ポートから読み込んだデータ **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); **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。
Nmc_WriteReg	ボードのライトレジスタ(WR0～WR7)にデータを書き込む。 **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); **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 **戻り値** なし **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);

関数名ロ	機能　及び　内容
Nmc_ReadReg	ボードのリードレジスタ(RR0～RR7)からデータを読み出す。 **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); **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。 **戻り値** リードレジスタから読み出したデータ **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); **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。

関数名	機能　及び　内容

Nmc_SetEvent	割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。
	
	VC　　BOOL Nmc_SetEvent(int No, LPTHREAD_START_ROUTINE UserThread, LPVOID lpParameter);
	VB.NET　Function Nmc_SetEvent(ByVal No As Integer, ByVal UserFunc As CallBack,
	ByVal param As Integer) As Integer
	C#　　bool MC8000P.Nmc_SetEvent(int No, UserThread Callback, int param);
	
	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)）
	[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。
	[VB.NET]
	UserFunc　ユーザーメソッド（デリゲート型）
	[VB.NET]
	param　割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Integer限定)を
	指定します。
	[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。
	
	戻り値
	[VC] 成功するとTRUE、失敗するとFALSE
	[VB.NET] 成功すると０以外、失敗すると０
	[C#] 成功するとtrue、失敗するとfalse
	
	使用例
	
	[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, 0x8000);　　//　ＩＣ－Ａの停止時割込み発生（全軸）
	（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, 0x8000);　　//　ＩＣ－Ａの停止時割込み発生（全軸）
	■割込みユーザー関数例
	DWORD WINAPI MC_EventFunc0(LPVOID lpParam)
	{
	long Rr3X, Rr3Y, Rr3Z, Rr3U;
	Nmc_ReadEvent(0, 0, &Rr3X,&Rr3Y,&Rr3Z,&Rr3U); // ﾎﾞｰﾄﾞ0,IC-AのRR3割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); // ﾎﾞｰﾄﾞ1,IC-AのRR3割込みデータ読み出し
	・・・・
	return 0;
	}
	[VB.NET]
	'コールバックメソッドのアドレス定義変数
	Public callbackFunc As CallBack
	
	' 割込みユーザーメソッド Event_Callback を指定、設定
	callbackFunc = AddressOf Event_Callback
	Nmc_SetEvent(No, callbackFunc, param)
	[C#]
	// 割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);
	// 割込みユーザーメソッドの設定
	MC8000P.Nmc_SetEvent(no, MC8000P.callback[0], param);

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_ResetEvent</td><td>
割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。

VC　　BOOL Nmc_ResetEvent(int No);
VB.NET　Function Nmc_ResetEvent(ByVal No As Integer) As Integer
C#　　bool MC8000P.Nmc_ResetEvent(int No);

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)）

戻り値
　[VC]　成功するとTRUE、失敗するとFALSE
　[VB.NET] 成功すると０以外、失敗すると０
　[C#]　成功するとtrue、失敗するとfalse

使用例
　[VC]　status = Nmc_ResetEvent(No);
　[VB.NET] status = Nmc_ResetEvent(No)
　[C#]　status = MC8000P.Nmc_ResetEvent(No);
</td></tr>
<tr><td>Nmc_ReadEvent</td><td>
割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）

VC　　BOOL Nmc_ReadEvent(int No, int IcNo, long* RrIrX, long* RrIrY, long* RrIrZ, long* RrIrU);
VB.NET　Function Nmc_ReadEvent(ByVal No As Integer, ByVal IcNo As Integer,
　　ByRef RrIrX As Integer, ByRef RrIrY As Integer,
　　ByRef RrIrZ As Integer, ByRef RrIrU As Integer) As Integer
C#　　bool MC8000P.Nmc_ReadEvent(int No, int IcNo, out int RrIrX, out int RrIrY, out int RrIrZ,
　　out int RrIrU);

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

戻り値
　[VC]　成功するとTRUE、失敗するとFALSE
　[VB.NET] 成功すると０以外、失敗すると０
　[C#]　成功するとtrue、失敗するとfalse

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
　[VB.NET]
　　Dim Rr3X,Rr3Y,Rr3Z,Rr3U As Integer
　　Nmc_ReadEvent(No, IcNo, Rr3X, Rr3Y, Rr3Z, Rr3U)
　[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);

注意
　ボードで割込が発生した直後ドライバ内でＲＲ３を読み出してしまうのでＲＲ３はクリアされて
　しまいます。割込み発生直後の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。
</td></tr>
</table>

関数名	機能　及び　内容
Nmc_Reset	ボードに搭載しているＩＣをリセットする。 **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); **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。 **戻り値** なし **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);
Nmc_Command	指定軸の命令を実行する。（WR0に指定軸の命令を書く） **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); **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 **戻り値** なし **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);
Nmc_Command_IP	補間命令を実行する。（WR0に補間命令を書く）　※MCX314As専用 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_WriteReg0	ＷＲＯ(コマンドレジスタ)にデータを書き込む。 **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); **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 **戻り値** なし **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);
Nmc_WriteReg1	ＷＲ１(モードレジスタ１)にデータを書き込む。 **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); **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 **戻り値** なし **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);
Nmc_WriteReg2	ＷＲ２(モードレジスタ２)にデータを書き込む。 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_WriteReg3	ＷＲ３(モードレジスタ３)にデータを書き込む。 **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); **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 **戻り値** なし **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);
Nmc_WriteReg4	ＷＲ４(アウトプットレジスタ)にデータを書き込む。 **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); **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 **戻り値** なし **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);
Nmc_WriteReg5	ＷＲ５にデータを書き込む。 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_WriteReg6	ＷＲ６(ライトデータレジスタ１)にデータを書き込む。 **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); **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 **戻り値** なし **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);
Nmc_WriteReg7	ＷＲ７(ライトデータレジスタ２)にデータを書き込む。 **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); **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 **戻り値** なし **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);
Nmc_ReadReg0	ＲＲ０(主ステータスレジスタ)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ０(主ステータスレジスタ)のデータ **使用例** [VC]　　Data = Nmc_ReadReg0(No, IcNo);　　// ＲＲ０のデータを読む [VB.NET]　Data = Nmc_ReadReg0(No, IcNo) [C#]　　Data = MC8000P.Nmc_ReadReg0(No, IcNo);

関数名	機能　及び　内容
Nmc_ReadReg1	ＲＲ１(ステータスレジスタ１)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ１(ステータスレジスタ１)のデータ **使用例** [VC]　　Data = Nmc_ReadReg1(No, IcNo, AXIS_X);　　// Ｘ軸のＲＲ１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);
Nmc_ReadReg2	ＲＲ２(ステータスレジスタ２)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ２(ステータスレジスタ２)のデータ **使用例** [VC]　　Data = Nmc_ReadReg2(No, IcNo, AXIS_Y);　　// Ｙ軸のＲＲ２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);
Nmc_ReadReg4	ＲＲ４(インプットレジスタ１)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ４(インプットレジスタ１)のデータ **使用例** [VC]　　Data = Nmc_ReadReg4(No, IcNo);　　// ＲＲ４のデータを読む [VB.NET]　Data = Nmc_ReadReg4(No, IcNo) [C#]　　Data = MC8000P.Nmc_ReadReg4(No, IcNo);

関数名	機能　及び　内容
Nmc_ReadReg5	ＲＲ５(インプットレジスタ２)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ５(インプットレジスタ２)のデータ **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);
Nmc_ReadReg6	ＲＲ６(リードデータレジスタ１)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ６(リードデータレジスタ１)のデータ **使用例** [VC]　　Data = Nmc_ReadReg6(No, IcNo);　　// ＲＲ６のデータを読む [VB.NET]　Data = Nmc_ReadReg6(No, IcNo) [C#]　　Data = MC8000P.Nmc_ReadReg6(No, int IcNo);
Nmc_ReadReg7	ＲＲ７(リードデータレジスタ２)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ７(リードデータレジスタ２)のデータ **使用例** [VC]　　Data = Nmc_ReadReg7(No, IcNo);　　// ＲＲ７のデータを読む [VB.NET]　Data = Nmc_ReadReg7(No, IcNo) [C#]　　Data = MC8000P.Nmc_ReadReg7(No, IcNo);

関数名	機能　及び　内容

Nmc_Range	レンジを設定する。
	
	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);
	
	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。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　設定するデータ
	
	戻り値
	なし
	
	使用例
	[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);

Nmc_Jerk	加速度増加率（加加速度）を設定する。 **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); **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。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　設定するデータ **戻り値** なし **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);
Nmc_Acc	加速度を設定する。 **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); **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。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　設定するデータ **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_Dec	減速度を設定する。 **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); **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 **戻り値** なし **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);
Nmc_StartSpd	初速度を設定する。 **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); **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 **戻り値** なし **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);
Nmc_Speed	ドライブ速度を設定する。 **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); **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 **戻り値** なし **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);

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_Pulse</td><td>出力パルス数、あるいは補間終点を設定する。(VC,C#専用)
※MC8082P/MC8082Pe、MC8022P、MC8042Pには補間機能がありません。

出力パルス数は、定量パルスドライブの総出力パルス数です。
直線補間、円弧補間ドライブの時は、各軸の終点を設定します。
終点座標は、現在位置に対する相対値を指定します。
出力パルス数は符号無し32ビット、補間終点は符号有り32ビットの値をセットして下さい。

VC　　void Nmc_Pulse(int No, int IcNo, int axis, long wdata);
VB.NET　使用できません
C#　　補間終点(P)設定の場合
　　void MC8000P.Nmc_Pulse(int No, int IcNo, AXIS axis, int wdata);
　出力パルス数設定の場合
　　void MC8000P.Nmc_Pulse(int No, int IcNo, AXIS axis, uint wdata);

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

戻り値
　なし

使用例
　[VC]　Nmc_Pulse(No, IcNo, AXIS_X, 2000);　　// 出力パルス数に 2000 を設定する(Ｘ軸)
　　Nmc_Pulse(No, IcNo, AXIS_Y, 300);　　// 補間終点に 300 を設定する(Ｙ軸)
　　Nmc_Pulse(No, IcNo, AXIS_Z, -400);　　// 補間終点に -400 を設定する(Ｚ軸)
　[C#]　MC8000P.Nmc_Pulse(No, IcNo, AXIS.X, 2000);
　　MC8000P.Nmc_Pulse(No, IcNo, AXIS.Y, 300);
　　MC8000P.Nmc_Pulse(No, IcNo, AXIS.Z, -400);</td></tr>
<tr><td>Nmc_Pulse_VB</td><td>出力パルス数、あるいは補間終点を設定する。(VB.NET専用)
※MC8082P/MC8082Pe、MC8022P、MC8042Pには補間機能がありません。

出力パルス数は、定量パルスドライブの総出力パルス数です。
直線補間、円弧補間ドライブの時は、各軸の終点を設定します。
終点座標は、現在位置に対する相対値を指定します。

VC　　使用できません
VB.NET　Sub Nmc_Pulse_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

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

戻り値
　なし

使用例
　[VB.NET]　Call Nmc_Pulse_VB(No, IcNo, AXIS_X, 2000)　' 出力パルス数に 2000 を設定する(Ｘ軸)
　　Call Nmc_Pulse_VB(No, IcNo, AXIS_Y, 300)　' 補間終点に 300 を設定する(Ｙ軸)
　　Call Nmc_Pulse_VB(No, IcNo, AXIS_Z, -400)　' 補間終点に -400 を設定する(Ｚ軸)</td></tr>
</table>

関数名	機能　及び　内容
Nmc_DecP	マニュアル減速点を設定する。(VC,C#専用) **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); **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 **戻り値** なし **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);
Nmc_DecP_VB	マニュアル減速点を設定する。(VB.NET専用) **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 **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 **戻り値** なし **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)
Nmc_Center	円弧中心点を設定する。 ※MC8043P/MC8043Pe専用 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_Lp	論理位置カウンタを設定する。 **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); **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 **戻り値** なし **使用例** [VC]　Nmc_Lp(No, IcNo, AXIS_ALL, 0);　// 全軸の論理位置カウンタをＯクリアする [VB.NET]　Call Nmc_Lp(No, IcNo, AXIS_ALL, 0) [C#]　MC8000P.Nmc_Lp(No, IcNo, AXIS.ALL, 0);
Nmc_Ep	実位置カウンタを設定する。 **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); **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 **戻り値** なし **使用例** [VC]　Nmc_Ep(No, IcNo, AXIS_ALL, 0);　// 全軸の実位置カウンタをＯクリアする [VB.NET]　Call Nmc_Ep(No, IcNo, AXIS_ALL, 0) [C#]　MC8000P.Nmc_Ep(No, IcNo, AXIS.ALL, 0);
Nmc_CompP	ＣＯＭＰ＋レジスタを設定する。 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_CompM	ＣＯＭＰ－レジスタを設定する。 **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); **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 **戻り値** なし **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);
Nmc_AccOfst	加速カウンタオフセットを設定する。 **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); **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 **戻り値** なし **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);
Nmc_DJerk	減速度増加率を設定する。　※MC8043P/MC8043Pe専用 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_HomeSpd	原点検出速度を設定する。 **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); **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 **戻り値** なし **使用例** [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);
Nmc_ExpMode	拡張モードを設定する。　※MC8043P/MC8043Pe専用 **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); **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。 EM6_data　拡張モードレジスタ EM6 に設定するデータ EM7_data　拡張モードレジスタ EM7 に設定するデータ **戻り値** なし **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ﾌﾟ1,2,4の実行を設定する [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);

関数名	機能　及び　内容
Nmc_SyncMode	同期動作モードを設定する。　※MC8043P/MC8043Pe専用 **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); **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 **戻り値** なし **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 [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);
Nmc_HomeMode	自動原点出しモードを設定する。　※MC8082P/MC8082Pe、MC8022P、MC8042P専用関数 **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); **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 **戻り値** なし **使用例**　Ｘ軸自動原点出しモードに、自動原点出しｽﾃｯﾌﾟ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);

関数名	機能　及び　内容
Nmc_ReadLp	論理位置カウンタを読み出す。 **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); **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。 **戻り値** 現在の論理位置カウンタの値 **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);
Nmc_ReadEp	実位置カウンタを読み出す。 **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); **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。 **戻り値** 現在の実位置カウンタの値 **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)
Nmc_ReadSpeed	現在ドライブ速度を読み出す。 **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); **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。 **戻り値** 現在ドライブ速度 **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);

関数名	機能　及び　内容
Nmc_ReadAccDec	現在加／減速度を読み出す。 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。 **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); **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。 **戻り値** 現在加／減速度 **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);
Nmc_ReadSyncBuff	同期バッファレジスタを読み出す。　※MC8043P/MC8043Pe専用 **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); **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。 **戻り値** 同期バッファレジスタの値 **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);

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_GetDriveStatus</td><td>
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。

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);

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。

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。

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″);
</td></tr>
<tr><td>Nmc_GetCNextStatus</td><td>
連続補間次データ書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。

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);

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。

戻り値
連続補間次データ書込み可能の場合は、０以外を返す。
連続補間次データ書込み可能ではない場合は、０を返す。

使用例
[VC]　If(Nmc_GetCNextStatus(No, IcNo) != 0)　// 次データ書込み可能の場合
AfxMessageBox(″連続補間次データ書込み可能である″);
else
AfxMessageBox(″連続補間次データ書込み可能ではない″);
[VB.NET]　If Nmc_GetCNextStatus(No, IcNo) <> 0 Then
Call MsgBox(″連続補間次データ書込み可能である″)
Else
Call MsgBox(″連続補間次データ書込み可能ではない″)
End If
[C#]　if(MC8000P.Nmc_GetCNextStatus(No, IcNo) != 0)
MessageBox.Show(″連続補間次データ書込み可能である″);
else
MessageBox.Show(″連続補間次データ書込み可能ではない″);
</td></tr>
</table>

関数名	機能　及び　内容
Nmc_GetBpSc	ＢＰ補間スタックカウンタの値を取得する。　※MC8043P/MC8043Pe専用 **VC**　int　Nmc_GetBpSc(int No, int IcNo); **VB.NET**　Function Nmc_GetBpSc(ByVal No As Integer, ByVal IcNo As Integer) As Integer **C#**　int　MC8000P.Nmc_GetBpSc(int No, int IcNo); **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。 **戻り値** 現在のビットパターン補間スタックカウンタの値 **使用例** [VC]　Data = Nmc_GetBpSc(No, IcNo);　// ＢＰ補間スタックカウンタの値を取得 [VB.NET]　Data = Nmc_GetBpSc(No, IcNo) [C#]　Data = MC8000P.Nmc_GetBpSc(No, IcNo);
Nmc_WriteRegSetAxis	指定軸の指定したライトレジスタ(ＷＲ１～ＷＲ３のいずれか)にデータを書き込む。 **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); **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。複数軸指定可能。詳細は「4.1.3 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]　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 wdata　書き込むデータ **戻り値** なし **使用例**　全軸のＷＲ２にALARM有効(2000)Hを書き込む [VC]　Nmc_WriteRegSetAxis(No, IcNo, AXIS_ALL, MCX_WR2, 0x2000); [VB.NET]　Call Nmc_WriteRegSetAxis(No, IcNo, AXIS_ALL, MCX_WR2, &H2000) [C#]　MC8000P.Nmc_WriteRegSetAxis(No, IcNo, AXIS.ALL, REG_MCX.WR2, 0x2000);

関数名	機能　及び　内容
Nmc_ReadRegSetAxis	指定軸の指定したリードレジスタ(ＲＲ１、ＲＲ２のどちらか)のデータを読み出す。 **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); **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。 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 **戻り値** 指定軸の指定したリードレジスタのデータ **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);
Nmc_WriteData	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) **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); **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。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((00)H～(0E)H, (61)H)。例：レンジ設定は(00)H。 ※MC8082P/MC8082Pe、MC8022P、MC8042Pは(08)H,(0E)Hを除く。 詳細は「4.1.3 補足説明」(1)参照。 wdata　書き込むデータ **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_WriteData2	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) **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); **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。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 **戻り値** なし **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);
Nmc_ReadData	データ読み出し命令を実行し、データを読み出す。 **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); **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。 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。 **戻り値** 読み出したデータ **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);

関数名	機能　及び　内容
Nmc_2BPExec	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。 **VC**　DWORD　Nmc_2BPExec(int No, int IcNo, DATA_2BP* pData2Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE); **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 **C#**　Nmc_Status MC8000P.Nmc_2BPExec(int No, int IcNo, DATA_2BP[] pData2Bp, int DataCnt, int IpAxis, bool ContinueFlg); **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 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のｱﾄﾞﾚｽ)。 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 **戻り値** 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。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード 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_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_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） **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(No, IcNo, Data2Bp, 2, 0x04);　// ２軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y軸 if(Ret == BP_END)　AfxMessageBox("正常終了");　// 戻り値正常 [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(No, IcNo, Data2Bp(0), 2, &H4, False) '２軸ＢＰ補間実行。ﾃﾞｰﾀ数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	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。

VC　DWORD　Nmc_3BPExec(int No, int IcNo, DATA_3BP* pData3Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE);

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

C#　Nmc_Status MC8000P.Nmc_3BPExec(int No, int IcNo, DATA_3BP[] pData3Bp, int DataCnt, int IpAxis, bool ContinueFlg);

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
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のｱﾄﾞﾚｽ)。 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

戻り値

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。

■正常終了

BP_END	BP補間処理正常終了

■エラーコード

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_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_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）

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(No, IcNo, Data3Bp, 2, 0x24); // ３軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y,Z軸

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

[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(No, IcNo, Data3Bp(0), 2, &H24, False) '３軸ＢＰ補間実行。ﾃﾞｰﾀ数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	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。 **VC**　　DWORD　Nmc_2BPExec_BG(HWND User_hWnd, int No, int IcNo, DATA_2BP* pData2Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE); **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 **C#**　　Nmc_Status MC8000P.Nmc_2BPExec_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_2BP[] pData2Bp, int DataCnt, int IpAxis, bool ContinueFlg); **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 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のｱﾄﾞﾚｽ)。 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 **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 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。 ■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 ■エラーコード(補間開始前のエラー) 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 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。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード(補間開始後のエラー) 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）

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。ﾃﾞｰﾀ数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。ﾃﾞｰﾀ数
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	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。 **VC**　　DWORD　Nmc_3BPExec_BG(HWND User_hWnd, int No, int IcNo, DATA_3BP* pData3Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE); **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 **C#**　　Nmc_Status MC8000P.Nmc_3BPExec_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_3BP[] pData3Bp, int DataCnt, int IpAxis, bool ContinueFlg); **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 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のｱﾄﾞﾚｽ)。 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 **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 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。 ■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 ■エラーコード(補間開始前のエラー) 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 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。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード(補間開始後のエラー) 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）

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。ﾃﾞｰﾀ数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。ﾃﾞｰﾀ数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	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 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。 **VC**　DWORD　Nmc_2CIPExec(int No, int IcNo, DATA_2CIP* pData2Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB.NET**　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 **C#**　Nmc_Status MC8000P. Nmc_2CIPExec(int No, int IcNo, DATA_2CIP[] pData2Cip, int DataCnt, int IpAxis, bool SpdChgFlg, bool ContinueFlg); **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 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のｱﾄﾞﾚｽ)。 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。ﾃﾞｰﾀ数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。ﾃﾞｰﾀ数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	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。 **VC**　DWORD　Nmc_3CIPExec(int No, int IcNo, DATA_3CIP* pData3Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **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 **C#**　Nmc_Status MC8000P.Nmc_3CIPExec(int No, IcNo, DATA_3CIP[] pData3Cip, int DataCnt, int IpAxis, bool SpdChgFlg, bool ContinueFlg); **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 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のｱﾄﾞﾚｽ)。 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。 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。 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_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。 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] 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。ﾃﾞｰﾀ数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。ﾃﾞｰﾀ数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	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。 **VC**　DWORD　Nmc_2CIPExec_BG(HWND User_hWnd, int No, int IcNo, DATA_2CIP* pData2Cip, int DataCnt,int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB.NET**　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 **C#**　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); **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のｱﾄﾞﾚｽ)。 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。ﾃﾞｰﾀ数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。ﾃﾞｰﾀ数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
      }
    }
```

関数名	機能　及び　内容
Nmc_3CIPExec_BG	指定した補間データで３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。 **VC**　DWORD　Nmc_3CIPExec_BG(HWND User_hWnd, int No, int IcNo, DATA_3CIP* pData3Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB.NET**　Function Nmc_3CIPExec_BG(ByVal User_hWnd As Integer, 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 **C#**　Nmc_Status MC8000P.Nmc_3CIPExec_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_3CIP[] pData3Cip, int IpAxis, bool SpdChgFlg, bool ContinueFlg); **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 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のｱﾄﾞﾚｽ)。 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。 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。 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_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。 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 ：変更する、false ：変更しない **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]

```
    {
       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。ﾃﾞｰﾀ数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。ﾃﾞｰﾀ数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	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。 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 **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); **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。 **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Ｏ以外、失敗するとＯ [C#]　補間処理の中断に成功するとtrue、失敗するとfalse **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);
Nmc_IPGetMsgNo	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 ◆ＢＰ補間　：　WM_BP_END ◆連続補間　：　WM_CIP_END **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) **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。 **戻り値** なし **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 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)

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 // ＷＲ０
   #define MCX_WR1        0x0001 // ＷＲ１
   #define MCX_WR2        0x0002 // ＷＲ２
   #define MCX_WR3        0x0003 // ＷＲ３
   #define MCX_WR4        0x0004 // ＷＲ４
   #define MCX_WR5        0x0005 // ＷＲ５
   #define MCX_WR6        0x0006 // ＷＲ６
   #define MCX_WR7        0x0007 // ＷＲ７

   #define MCX_RR0        0x0000 // ＲＲ０
   #define MCX_RR1        0x0001 // ＲＲ１
   #define MCX_RR2        0x0002 // ＲＲ２
   #define MCX_RR3        0x0003 // ＲＲ３
   #define MCX_RR4        0x0004 // ＲＲ４
   #define MCX_RR5        0x0005 // ＲＲ５
   #define MCX_RR6        0x0006 // ＲＲ６
   #define MCX_RR7        0x0007 // ＲＲ７
```

[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、ボ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_DFREE          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              // ＢＰレジスタ書き込み可
#define CMD_BP_DISABLED           0x37              // ＢＰレジスタ書き込み不可
#define CMD_BP_STACK              0x38              // ＢＰデータスタック
#define CMD_BP_CLR                0x39              // ＢＰ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    // ＢＰレジスタ書き込み可
#define MC8000P_CMD_BPDISABLED    CMD_BP_DISABLED   // ＢＰレジスタ書き込み不可
#define MC8000P_CMD_BPSTACK       CMD_BP_STACK      // ＢＰデータスタック
#define MC8000P_CMD_BPCLR         CMD_BP_CLR        // ＢＰ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_DFREE            = 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_BPIP2             = CMD_IP_2BP,    // 2軸ビットパターン補間ドライブ
  MC8000P_CMD_BPIP3             = CMD_IP_3BP,    // 3軸ビットパターン補間ドライブ
  MC8000P_CMD_BPENABLED         = CMD_BP_ENABLED,    // ＢＰレジスタ書き込み可
  MC8000P_CMD_BPDISABLED        = CMD_BP_DISABLED,   // ＢＰレジスタ書き込み不可
  MC8000P_CMD_BPSTACK           = CMD_BP_STACK,      // ＢＰデータスタック
  MC8000P_CMD_BPCLR             = CMD_BP_CLR,        // ＢＰデータクリア
  MC8000P_CMD_IP1STEP           = CMD_IP_1STEP,      // 補間シングルステップ
  MC8000P_CMD_DECEN             = CMD_IP_DEC_VALID,  // 減速有効
  MC8000P_CMD_DECDIS            = CMD_IP_DEC_INVALID,// 減速無効
  MC8000P_CMD_CLRINTRPT         = 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ＢＰ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) ＢＰ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ＢＰ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ＢＰ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

⑦ＩＣ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）ＩＣ　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

①１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);

②２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);

③３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

①ＶＣ

```
// 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;
```

```
  // 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
```

②ＶＢ．ＮＥＴ

```
’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ビットデータの下位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
										第３軸		第２軸		第１軸	

◆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	００
Y	０１
Z	１０
U	１１

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。

①ＩＣを１つ搭載しているボードの場合は０を指定します。

②ＩＣ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。

④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。

⑤割込みを行う場合は、割込みユーザー関数（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にて割込みを処理する方法

①Nmc_Open関数にて割込みを使用する設定にします。

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

②Nmc_SetEvent関数にて割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の停止時割込み発生（全軸）
```

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

■割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);
```

■ＶＢ．ＮＥＴにて割込みを処理する方法
ＶＣと同じ手順で処理をします。

■Ｃ＃にて割込みを処理する方法

①MC8000P.Nmc_SetEventにて、割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, 0x8000);           // 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。RR3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 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　　※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。

■ＢＰ補間（ビットパターン補間）について　　※MC8043P/MC8043Peのみ

ＢＰ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、ＢＰ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ＢＰ補間関数を使用してＢＰ補間の処理を行うこともできます。但し、ＢＰ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

ＢＰ補間関数を使用する場合の注意事項：
ＢＰ補間関数は、次のＢＰデータを書き込み、補間命令書き込み後、エラーチェック行い、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ＢＰデータを書き込みます。本関数は、ＢＰ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、ＢＰ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ＢＰ補間の処理を作成してください。また、ＢＰ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
②ＷＲ５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、ＷＲ６，ＷＲ７にデータを書き込む処理、ＲＲ６，ＲＲ７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

◆ＷＲ６，ＷＲ７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

◆ＲＲ６，ＲＲ７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

ＷＲ１～ＷＲ３書き込み、ＲＲ１～ＲＲ２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。

◆ＷＲ１～ＷＲ３書き込み
Nmc_WriteReg1　Nmc_WriteReg2　Nmc_WriteReg3　Nmc_WriteRegSetAxis

◆ＲＲ１～ＲＲ２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

ＷＲ１～ＷＲ３書き込み、ＲＲ１～ＲＲ２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、ＷＲ１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);         // ＷＲ１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);    // ＷＲ１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ＷＲ１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 );          // ＷＲ６書き込み（ＩＣ－Ａ）
②  Nmc_OutPort( No, MCX_WR0, 0x0105 );        // ＷＲ６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);             // ＷＲ６書き込み
④  Nmc_Command(No, IcNo, AXIS_X, 0x05);       // ＷＲ６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、ＷＲ６，ＷＲ７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ＲＲ６，ＲＲ７に読み出す（ＩＣ－Ａ）
②  d6 = Nmc_InPort( No, MCX_RR6 );     // ＲＲ６を読み出す（ＩＣ－Ａ）
③  d7 = Nmc_InPort( No, MCX_RR7 );     // ＲＲ７を読み出す（ＩＣ－Ａ）
```

この場合、①と②の間、②と③の間に、ＲＲ６，ＲＲ７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：512μsec

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)に適正値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：512μsec

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。

(例)
```
Nmc_WriteReg3(0, 0, AXIS_ALL, 0x4F00);        // ＩＣ－Ａの設定
Nmc_WriteReg3(0, 1, AXIS_ALL, 0x4F00);        // ＩＣ－Ｂの設定（複数ＩＣ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

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）
・ＲＲ３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）
・ＲＲ３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

①　ＲＲ３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の割り込み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
①　Nmc_ReadEventにてRR3の割り込み要因を読み出し、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。

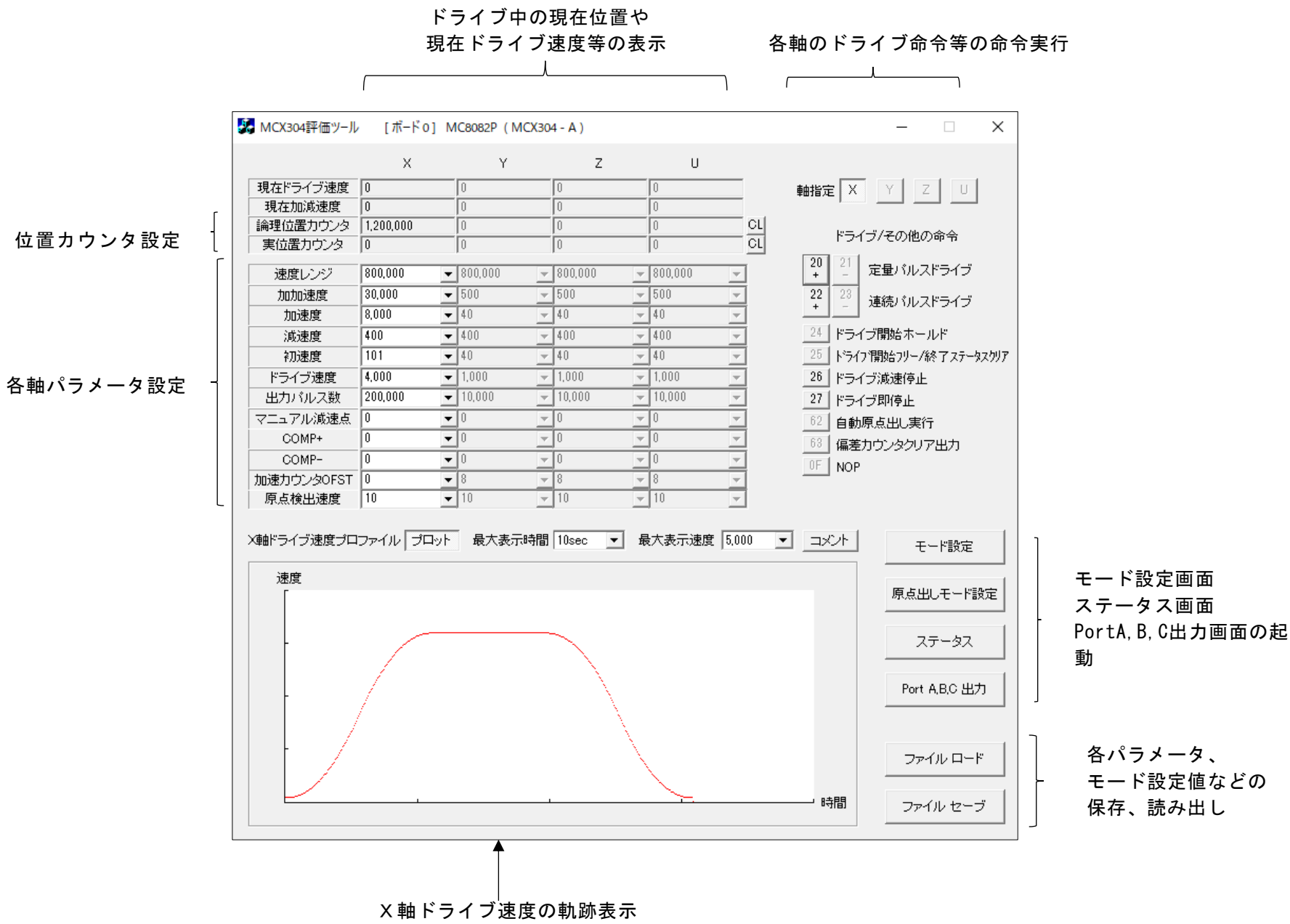
4.3.2　機能概要

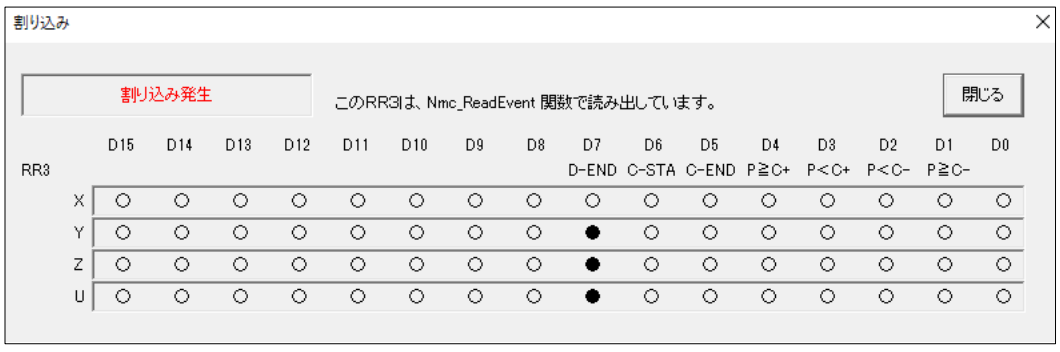
評価ツールを起動すると、ボード番号選択画面が表示され、ボード番号（ボードのロータリースイッチ番号(0～F)）を選択することができます。ボード名表示ボタンを押すと、ボード番号の横にボード名(本ドライバを使用しているボードのみ)を表示することができます。ボード番号を選択後、ＯＫボタンを押すと、メイン画面が表示されます。

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

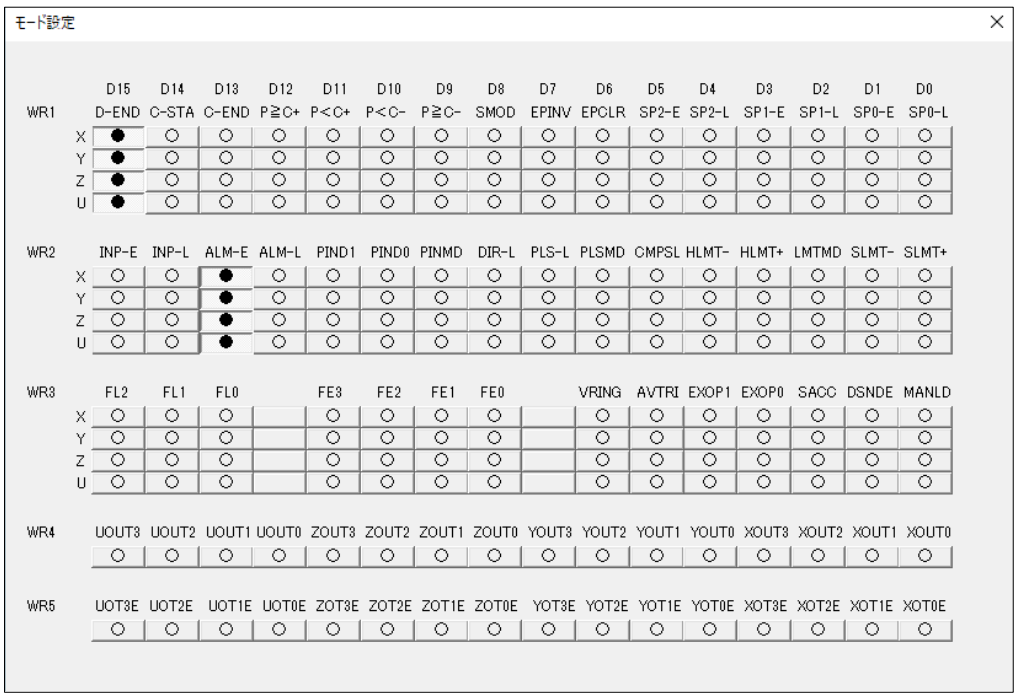
メイン画面では、下図のような操作を行います。





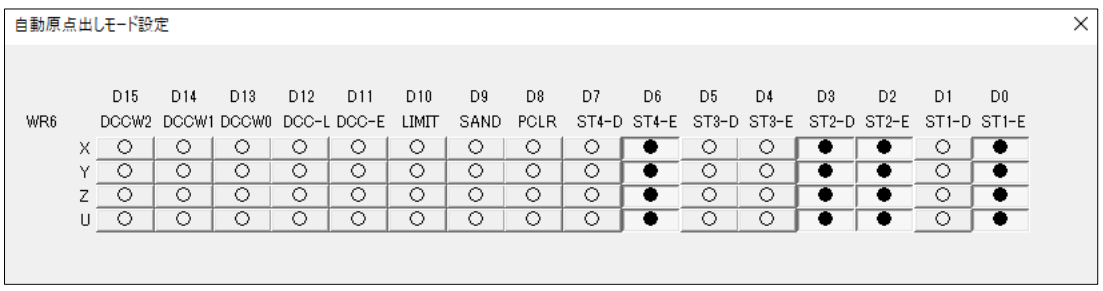
4.3.4　モード設定画面

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。



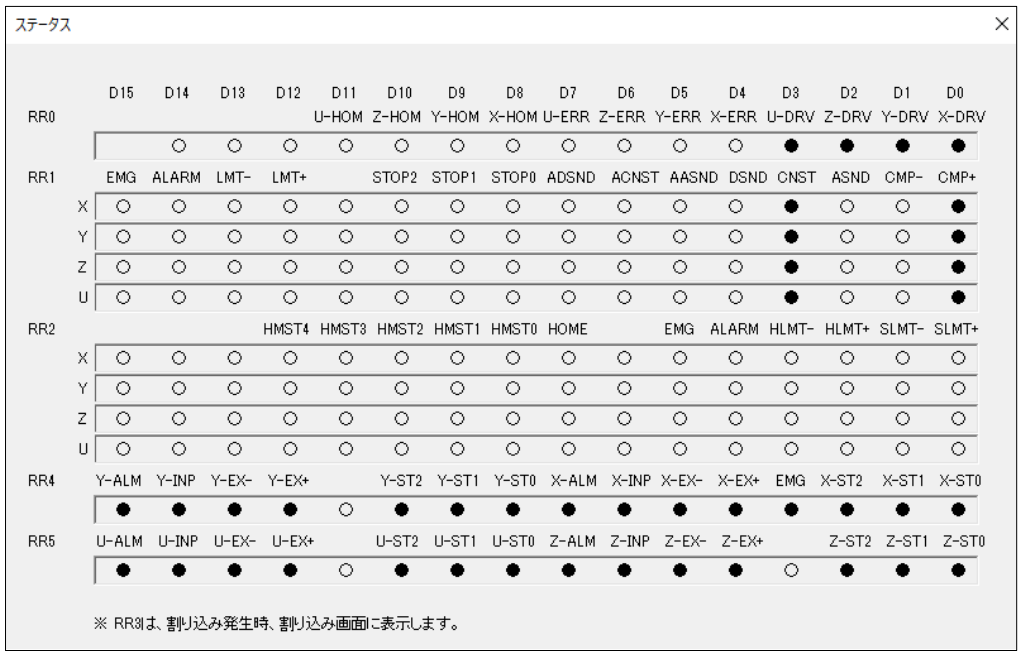
4.3.5　自動原点出しモード設定画面

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。



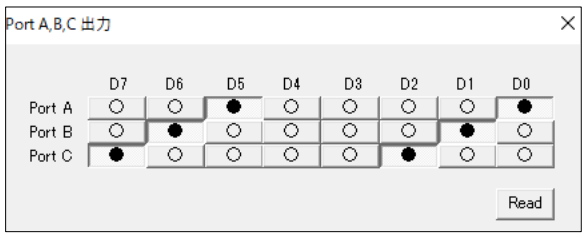
4.3.6　ステータス画面

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。



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。



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。

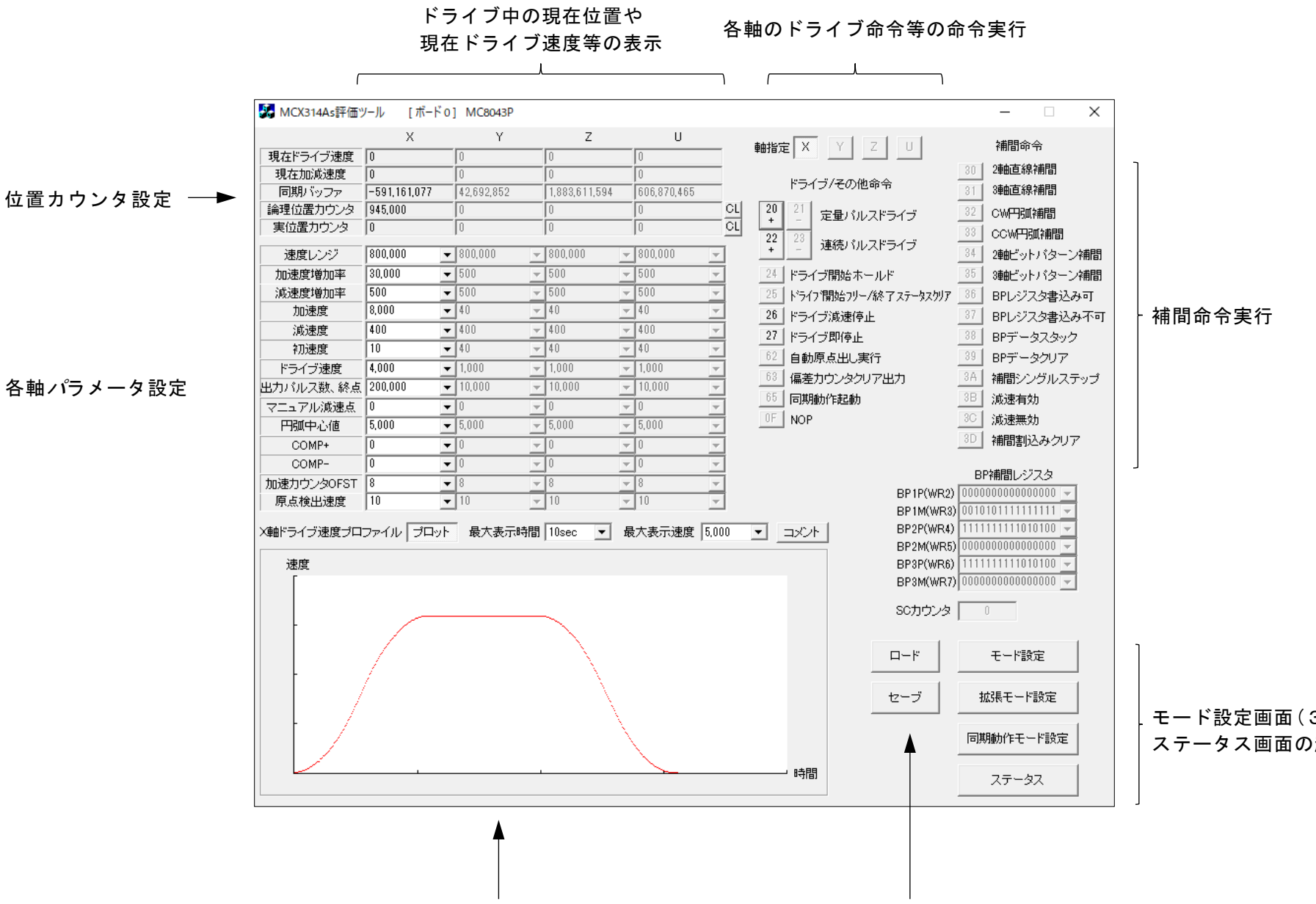
4.4.2　機能概要

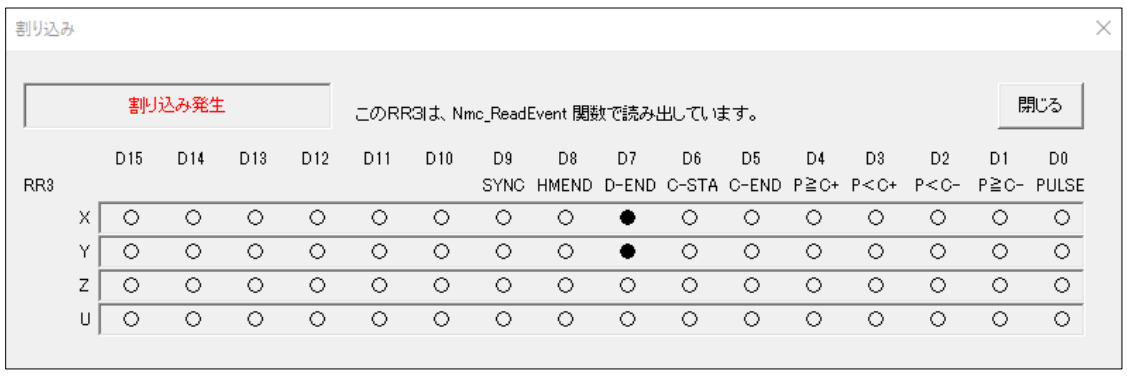
評価ツールを起動すると、ボード番号選択画面が表示され、ボード番号（ボードのロータリースイッチ番号(0～F)）を選択することができます。ボード名表示ボタンを押すと、ボード番号の横にボード名(本ドライバを使用しているボードのみ)を表示することができます。ボード番号を選択後、ＯＫボタンを押すと、メイン画面が表示されます。

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。

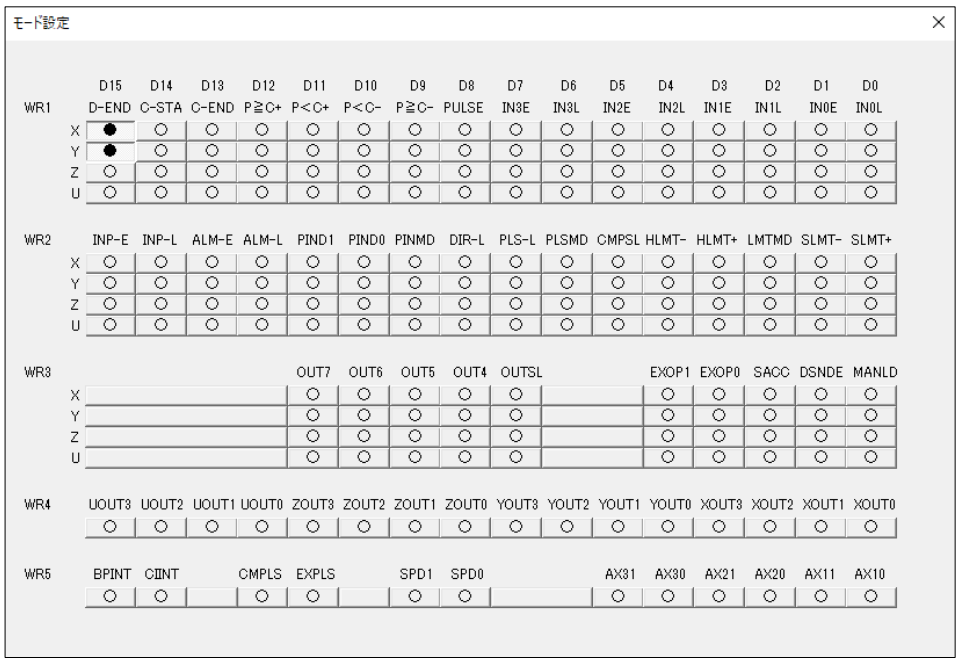




割り込み発生時に
割り込み画面表示

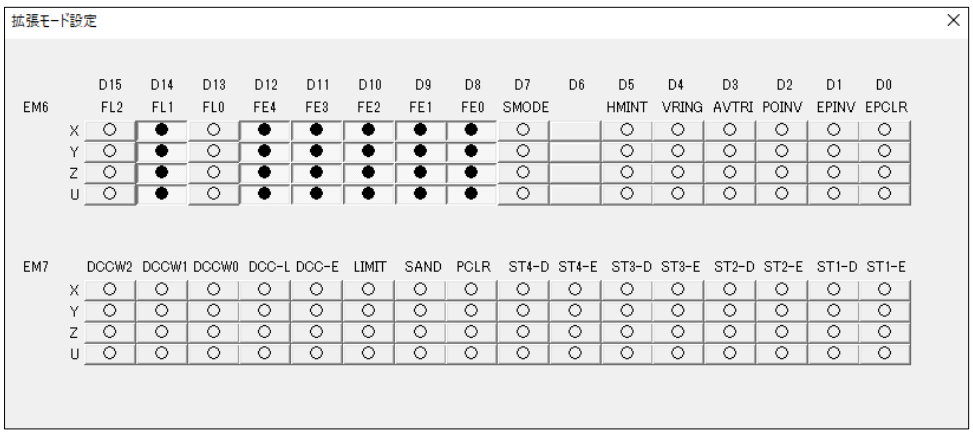
4.4.4　モード設定画面

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。



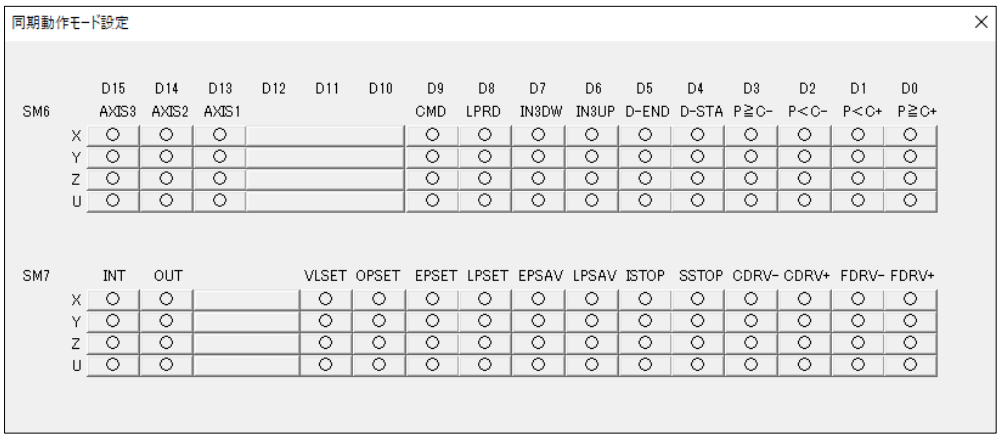
4.4.5　拡張モード設定画面

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。



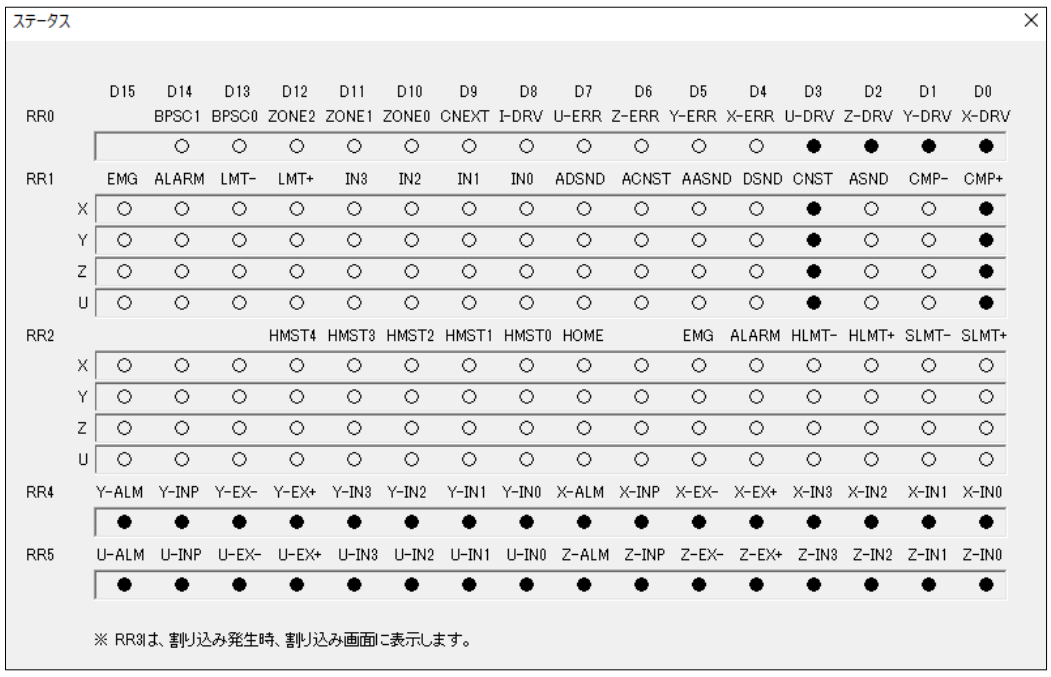
4.4.6　同期動作モード設定画面

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。



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。



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ＩＣ	ＩＣ数	軸数
MC8541P / MC8541Pe	MCX514	1	4
MC8581P / MC8581Pe	MCX514	2	8
MC8543PeL	MCX514	1	4

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が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	○	○	○	102	
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	○	○	○	103	
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	○	○	○	104	
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	○	○	○	105	
Nmc_ResetEvent	割込みを処理するユーザー関数の設定を解除する	○	○	○	106	
Nmc_ReadEvent	割込み発生直後の各軸ＲＲ１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	○	○	○	107	
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	○	○	○	108	

（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	○	○	○	108	
Nmc_WriteReg1	ＷＲ１(モードレジスタ１)　　書き込み	○	○	○	109	
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	○	○	○	110	
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	○	○	○	111	

（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	○	○	○	111	
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)	○	○	○	112	
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	○	○	○	113	
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）	○	○	○	114	
Nmc_DJerk	減速度増加率設定	○	○	○	〃	
Nmc_Acc	加速度設定	○	○	○	〃	
Nmc_Dec	減速度設定	○	○	○	115	
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)	○	×	○	116	
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)	○	×	○	117	
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	○	○	○	118	
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	ソフトリミットーレジスタ設定	○	○	○	119	
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	○	○	○	120	
Nmc_LpMax	論理位置カウンタ最大値設定	○	○	○	〃	
Nmc_RpMax	実位位置カウンタ最大値設定	○	○	○	〃	
Nmc_MR0	多目的レジスタ０設定	○	○	○	121	
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	○	○	○	122	
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	○	○	○	123	
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	○	○	○	124	
Nmc_PIO1Mode	ＰＩＯ信号設定１	○	○	○	〃	
Nmc_PIO2Mode	ＰＩＯ信号設定２･その他設定	○	○	○	〃	
Nmc_HMSrch1Mode	自動原点出しモード設定１	○	○	○	125	
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	○	○	○	126	
Nmc_Sync0Mode	同期動作ＳＹＮＣ０設定	○	○	○	〃	
Nmc_Sync1Mode	同期動作ＳＹＮＣ１設定	○	○	○	〃	
Nmc_Sync2Mode	同期動作ＳＹＮＣ２設定	○	○	○	127	
Nmc_Sync3Mode	同期動作ＳＹＮＣ３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	○	○	○	128	
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	○	○	○	129	
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	○	○	○	130	
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	○	○	○	131	
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	○	○	○	132	
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	○	○	○	133	
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	○	○	○	134	
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	○	○	○	135	

（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	○	○	○	135	
Nmc_GetCNextStatus	連続補間次データ書込み可能状態取得	○	○	○	136	
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)	○	○	○	137	
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）	○	○	○	138	
Nmc_ReadData	データ読み出し(4バイト読み出し)	○	○	○	〃	

（１０）ＢＰ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ＢＰ補間実行	○	○	○	139	
Nmc_3BPExecMC8500P	３軸ＢＰ補間実行	○	○	○	141	
Nmc_4BPExecMC8500P	４軸ＢＰ補間実行	○	○	○	143	
Nmc_2BPExecMC8500P_BG	２軸ＢＰ補間実行（バックグラウンドで実行）	○	○	○	145	
Nmc_3BPExecMC8500P_BG	３軸ＢＰ補間実行（バックグラウンドで実行）	○	○	○	148	
Nmc_4BPExecMC8500P_BG	４軸ＢＰ補間実行（バックグラウンドで実行）	○	○	○	151	
Nmc_2CIPExecMC8500P	２軸連続補間実行	○	○	○	154	
Nmc_3CIPExecMC8500P	３軸連続補間実行	○	○	○	156	
Nmc_4CIPExecMC8500P	４軸連続補間実行	○	○	○	158	
Nmc_2CIPExecMC8500P_BG	２軸連続補間実行（バックグラウンドで実行）	○	○	○	160	
Nmc_3CIPExecMC8500P_BG	３軸連続補間実行（バックグラウンドで実行）	○	○	○	163	
Nmc_4CIPExecMC8500P_BG	４軸連続補間実行（バックグラウンドで実行）	○	○	○	166	
Nmc_IPStop	補間実行を中断する	○	○	○	169	
Nmc_IPGetMsgNo	補間終了時の受信ﾒｯｾｰｼﾞからﾎﾞｰﾄﾞ番号と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	○	○	○	170	
Nmc_HLExec	ヘリカル補間実行	○	○	○	172	

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	ボードの使用を開始する。 **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); **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。 **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 **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);
Nmc_Close	ボードの使用を終了する。 **VC**　　BOOL　　Nmc_Close(int No); **VB.NET**　Function Nmc_Close(ByVal No As Integer) As Integer **C#**　　bool　　MC8000P.Nmc_Close(int No); **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)） **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 **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);
Nmc_CloseAll	全てのボードの使用を終了する。 **VC**　　BOOL　　Nmc_CloseAll(void); **VB.NET**　Function Nmc_CloseAll() As Integer **C#**　　bool　　MC8000P.Nmc_CloseAll(); **入力パラメータ** 　なし **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 **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();

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_GetBoardInfo</td><td>
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。

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);

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。

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Ｏ以外、失敗するとＯ
　[C#]　　取得が成功するとtrue、失敗するとfalse

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)
</td></tr>
<tr><td>Nmc_OutPort</td><td>
出力ポートに２バイトデータを書き込む。

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);

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

戻り値
　なし

使用例
　[VC]　　Nmc_OutPort(No, 0, MC8500P_CMD_RST);　// ＩＣ－Ａのコマンドリセット（WR0書き込み）
　[VB.NET]　Call Nmc_OutPort(No, 0, MC8500P_CMD_RST)
　[C#]　　MC8000P.Nmc_OutPort(No, REG_MCX.WR0_A, MC8500P_CMD_RST);
</td></tr>
<tr><td>Nmc_InPort</td><td>
入力ポートから２バイトデータを読み出す。

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);

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。

戻り値
　入力ポートから読み込んだデータ

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);

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。
</td></tr>
</table>

関数名	機能　及び　内容
Nmc_WriteReg	ボードのライトレジスタ(WR0～WR7)にデータを書き込む。 **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); **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。 　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。 　Dat　　書き込むデータ **戻り値** 　なし **使用例** 　[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);
Nmc_ReadReg	ボードのリードレジスタ(RR0～RR7)からデータを読み出す。 **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); **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。 　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。 **戻り値** 　リードレジスタから読み出したデータ **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); **注意** 　[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。

関数名	機能　及び　内容

Nmc_SetEvent	割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。
	
	VC　　BOOL Nmc_SetEvent(int No, LPTHREAD_START_ROUTINE UserFunc, LPVOID lpParameter);
	VB.NET　Function Nmc_SetEvent(ByVal No As Integer, ByVal UserFunc As CallBack,
	ByVal param As Integer) As Integer
	C#　　bool MC8000P.Nmc_SetEvent(int No, UserThread Callback, int param);
	
	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)）
	[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。
	[VB.NET]
	UserFunc　ユーザーメソッド（デリゲート型）
	[VB.NET]
	param　割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Integer限定)を指定します。
	[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。
	
	戻り値
	[VC] 成功するとTRUE、失敗するとFALSE
	[VB.NET] 成功すると０以外、失敗すると０
	[C#] 成功するとtrue、失敗するとfalse
	
	使用例
	[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のドライブ終了割込み発生（全軸）
	
	（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のドライブ終了割込み発生（全軸）
	
	■割込みユーザー関数例
	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割込みデータ読み出し
	・・・・
	return 0;
	}
	[VB.NET]
	'コールバックメソッドのアドレス定義変数
	Public callbackFunc As CallBack
	
	' 割込みユーザーメソッド Event_Callback を指定、設定
	callbackFunc = AddressOf Event_Callback
	Nmc_SetEvent(No, callbackFunc, param)
	[C#]
	// 割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);
	// 割込みユーザーメソッドの設定
	MC8000P.Nmc_SetEvent(no, MC8000P.callback[0], param);

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_ResetEvent</td><td>割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。

VC　　BOOL Nmc_ResetEvent(int No);
VB.NET　Function Nmc_ResetEvent(ByVal No As Integer) As Integer
C#　　bool MC8000P.Nmc_ResetEvent(int No);

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)）

戻り値
　[VC]　成功するとTRUE、失敗するとFALSE
　[VB.NET] 成功すると０以外、失敗すると０
　[C#]　成功するとtrue、失敗するとfalse

使用例
　[VC]　status = Nmc_ResetEvent(No);
　[VB.NET] status = Nmc_ResetEvent(No)
　[C#]　status = MC8000P.Nmc_ResetEvent(No);</td></tr>
<tr><td>Nmc_ReadEvent</td><td>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）

VC　　BOOL Nmc_ReadEvent(int No, int IcNo, long* RrIrX, long* RrIrY, long* RrIrZ, long* RrIrU);
VB.NET　Function Nmc_ReadEvent(ByVal No As Integer, ByVal IcNo As Integer,
　　ByRef RrIrX As Integer, ByRef RrIrY As Integer,
　　ByRef RrIrZ As Integer, ByRef RrIrU As Integer) As Integer
C#　　bool MC8000P.Nmc_ReadEvent(int No, int IcNo, out int RrIrX, out int RrIrY, out int RrIrZ,
　　out int RrIrU);

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

戻り値
　[VC]　成功するとTRUE、失敗するとFALSE
　[VB.NET] 成功すると０以外、失敗すると０
　[C#]　成功するとtrue、失敗するとfalse

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
　[VB.NET]
　　Dim Rr1X,Rr1Y,Rr1Z,Rr1U As Integer
　　Nmc_ReadEvent(No, IcNo, Rr1X, Rr1Y, Rr1Z, Rr1U)
　[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);

注意
　ボードで割込が発生した直後ドライバ内でＲＲ１を読み出してしまうので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。</td></tr>
</table>

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_Reset</td><td>ボードに搭載しているＩＣをリセットする。

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);

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。

戻り値
　なし

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);</td></tr>
<tr><td>Nmc_Command</td><td>指定軸の命令を実行する。（WR0に指定軸の命令を書く）

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);

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　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MC850OP_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

戻り値
なし

使用例
[VC]　　Nmc_Command(No, IcNo, AXIS_X, MC850OP_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, MC850OP_CMD_DRVRL)
[C#]　　MC8000P.Nmc_Command(No, IcNo, AXIS.X, CMD.MC850OP_CMD_DRVRL);</td></tr>
</table>

関数名	機能　及び　内容
Nmc_Command_IP	補間命令を実行する。（WR0に補間命令を書く） **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); **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。 　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 **戻り値** 　なし **使用例** 　[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);
Nmc_WriteReg0	ＷＲＯ(コマンドレジスタ)にデータを書き込む。 **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); **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 **戻り値** 　なし **使用例** 　[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);

関数名	機能　及び　内容
Nmc_WriteReg1	ＷＲ１(モードレジスタ１)にデータを書き込む。 **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); **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 **戻り値** なし **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);
Nmc_WriteReg2	ＷＲ２(モードレジスタ２)にデータを書き込む。 **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); **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 **戻り値** なし **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);
Nmc_WriteReg3	ＷＲ３(モードレジスタ３)にデータを書き込む。 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_WriteReg4	ＷＲ４(アウトプットレジスタ１)にデータを書き込む。 **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); **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 **戻り値** 　なし **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);
Nmc_WriteReg5	ＷＲ５(アウトプットレジスタ２)にデータを書き込む。 **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); **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 **戻り値** 　なし **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);
Nmc_WriteReg6	ＷＲ６(ライトデータレジスタ１)にデータを書き込む。 **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); 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 戻り値 　なし 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);

関数名	機能　及び　内容
Nmc_WriteReg7	ＷＲ７(ライトデータレジスタ２)にデータを書き込む。 **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); **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 **戻り値** なし **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);
Nmc_ReadReg0	ＲＲ０(主ステータスレジスタ)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ０(主ステータスレジスタ)のデータ **使用例** [VC]　　Data = Nmc_ReadReg0(No, IcNo);　　// ＲＲ０のデータを読む [VB.NET]　Data = Nmc_ReadReg0(No, IcNo) [C#]　　Data = MC8000P.Nmc_ReadReg0(No, IcNo);
Nmc_ReadReg2	ＲＲ２(ステータスレジスタ２)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ２(ステータスレジスタ２)のデータ **使用例** [VC]　　Data = Nmc_ReadReg2(No, IcNo, AXIS_Y);　　// Ｙ軸のＲＲ２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);

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_ReadReg3</td><td>ＲＲ３(ステータスレジスタ３)のデータを読み出す。

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);

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。

戻り値
ＲＲ３(ステータスレジスタ３)のデータ

使用例
[VC]　Nmc_Command(No, IcNo, AXIS_Y, MC8500P_CMD_RR3P0);　// Ｙ軸のＲＲ３のページ０表示
Data = Nmc_ReadReg3(No, IcNo, AXIS_Y);　// Ｙ軸のＲＲ３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);

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。</td></tr>
<tr><td>Nmc_ReadReg3P</td><td>ＲＲ３(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)

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);

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。

戻り値
指定したページのＲＲ３(ステータスレジスタ３)データ

使用例
[VC]　Data = Nmc_ReadReg3P(No, IcNo, AXIS_Y, 0);　// Ｙ軸のＲＲ３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);</td></tr>
<tr><td>Nmc_ReadReg4</td><td>ＲＲ４(ＰＩＯリードレジスタ１)のデータを読み出す。

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);

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。

戻り値
ＲＲ４(ＰＩＯリードレジスタ１)のデータ

使用例
[VC]　Data = Nmc_ReadReg4(No, IcNo);　// ＲＲ４のデータを読む
[VB.NET]　Data = Nmc_ReadReg4(No, IcNo)
[C#]　Data = MC8000P.Nmc_ReadReg4(No, IcNo);</td></tr>
</table>

関数名	機能　及び　内容
Nmc_ReadReg5	ＲＲ５(ＰＩＯリードレジスタ２)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ５(ＰＩＯリードレジスタ２)のデータ **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);
Nmc_ReadReg6	ＲＲ６(リードデータレジスタ１)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ６(リードデータレジスタ１)のデータ **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);
Nmc_ReadReg7	ＲＲ７(リードデータレジスタ２)のデータを読み出す。 **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); **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。 **戻り値** ＲＲ７(リードデータレジスタ２)のデータ **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);

関数名	機能　及び　内容
Nmc_Jerk	加速度増加率を設定する。 **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); **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 **戻り値** なし **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);
Nmc_DJerk	減速度増加率を設定する。 **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); **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 **戻り値** なし **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);
Nmc_Acc	加速度を設定する。 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_Dec	減速度を設定する。 **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); **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 **戻り値** なし **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);
Nmc_StartSpd	初速度を設定する。 **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); **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 **戻り値** なし **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);
Nmc_Speed	ドライブ速度を設定する。 **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); **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 **戻り値** なし **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);

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_Pulse</td><td>出力パルス数、あるいは補間終点を設定する。(VC,C#専用)
出力パルス数は、定量パルスドライブの総出力パルス数です。
直線補間、円弧補間ドライブの時は、各軸の終点を設定します。
ヘリカル補間ドライブ時は、Ｚ、Ｕ軸の送り量を設定します。
終点座標は、現在位置に対する相対値を指定します。

VC　　void Nmc_Pulse(int No, int IcNo, int axis, long wdata);
VB.NET　使用できません
C#　　void MC8000P.Nmc_Pulse(int No, int IcNo, AXIS axis, int wdata);

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

戻り値
なし

使用例
[VC]　Nmc_Pulse(No, IcNo, AXIS_X, 2000);　　// 出力パルス数に 2000 を設定する(Ｘ軸)
Nmc_Pulse(No, IcNo, AXIS_Y, 300);　　// 補間終点に 300 を設定する(Ｙ軸)
Nmc_Pulse(No, IcNo, AXIS_Z, -400);　　// 補間終点に -400 を設定する(Ｚ軸)
[C#]　MC8000P.Nmc_Pulse(No, IcNo, AXIS.X, 2000);
MC8000P.Nmc_Pulse(No, IcNo, AXIS.Y, 300);
MC8000P.Nmc_Pulse(No, IcNo, AXIS.Z, -400);</td></tr>
<tr><td>Nmc_Pulse_VB</td><td>出力パルス数、あるいは補間終点を設定する。(VB.NET専用)
出力パルス数は、定量パルスドライブの総出力パルス数です。
直線補間、円弧補間ドライブの時は、各軸の終点を設定します。
ヘリカル補間ドライブ時は、Ｚ、Ｕ軸の送り量を設定します。
終点座標は、現在位置に対する相対値を指定します。

VC　　使用できません
VB.NET　Sub Nmc_Pulse_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

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「4.1.3 補足説明」(2)参照。
wdata　設定するデータ

戻り値
なし

使用例
[VB.NET]　Call Nmc_Pulse_VB(No, IcNo, AXIS_X, 2000)　' 出力パルス数に 2000 を設定する(Ｘ軸)

Call Nmc_Pulse_VB(No, IcNo, AXIS_Y, 300)　' 補間終点に 300 を設定する(Ｙ軸)
Call Nmc_Pulse_VB(No, IcNo, AXIS_Z, -400)　' 補間終点に -400 を設定する(Ｚ軸)</td></tr>
</table>

関数名	機能　及び　内容
Nmc_DecP	マニュアル減速点を設定する。(VC,C#専用) **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); **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 **戻り値** なし **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);
Nmc_DecP_VB	マニュアル減速点を設定する。(VB.NET専用) **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 **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 **戻り値** なし **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)
Nmc_Center	円弧中心点を設定する。 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_Lp	論理位置カウンタを設定する。 **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); **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 **戻り値** なし **使用例** [VC]　　Nmc_Lp(No, IcNo, AXIS_ALL, 0);　　// 全軸の論理位置カウンタをＯクリアする [VB.NET]　Call Nmc_Lp(No, IcNo, AXIS_ALL, 0) [C#]　　MC8000P.Nmc_Lp(No, IcNo, AXIS.ALL, 0);
Nmc_Ep	実位置カウンタを設定する。 **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); **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 **戻り値** なし **使用例** [VC]　　Nmc_Ep(No, IcNo, AXIS_ALL, 0);　　// 全軸の実位置カウンタをＯクリアする [VB.NET]　Call Nmc_Ep(No, IcNo, AXIS_ALL, 0) [C#]　　MC8000P.Nmc_Ep(No, IcNo, AXIS.ALL, 0);

Nmc_CompP	ソフトリミット＋レジスタを設定する。
	
	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);
	
	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
	
	戻り値
	なし
	
	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(Ｘ軸)
	[VB.NET]　Call Nmc_CompP(No, IcNo, AXIS_X, 50000)
	[C#]　　MC8000P.Nmc_CompP(No, IcNo, AXIS.X, 50000);

関数名	機能　及び　内容

Nmc_CompM	ソフトリミットーレジスタを設定する。
	
	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);
	
	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
	
	戻り値
	なし
	
	使用例
	[VC]　　Nmc_CompM(No, IcNo, AXIS_X, -50000); // ソフトリミットーレジスタに -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);

Nmc_AccOfst	加速カウンタオフセットを設定する。 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_HomeSpd	原点検出速度を設定する。 **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); **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 **戻り値** なし **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);

Nmc_LpMax	論理位置カウンタ最大値を設定する。
	
	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);
	
	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
	
	戻り値
	なし
	
	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);

Nmc_RpMax	実位置カウンタ最大値を設定する。
	
	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);
	
	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
	
	戻り値
	なし
	
	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);

関数名	機能　及び　内容
Nmc_MR0	多目的レジスタ０を設定する。 **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); **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 **戻り値** なし **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);
Nmc_MR1	多目的レジスタ１を設定する。 **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); **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 **戻り値** なし **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);
Nmc_MR2	多目的レジスタ２を設定する。 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容

Nmc_MR3	多目的レジスタ３を設定する。
	
	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);
	
	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
	
	戻り値
	なし
	
	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);

Nmc_SpeedInc	速度増減値を設定する。 **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); **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 **戻り値** なし **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);
Nmc_Timer	タイマー値を設定する。 **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); **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 **戻り値** なし **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);

関数名	機能　及び　内容
Nmc_TPMax	補間・終点最大値を設定する。 **VC**　　void Nmc_TPMax(int No, int IcNo, long wdata); **VB.NET**　Sub　Nmc_TPMax(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) **C#**　　void MC8000P.Nmc_TPMax(int No, int IcNo, int wdata); **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 **戻り値** 　なし **使用例** 　[VC]　　Nmc_TPMax(No, IcNo, 10000);　　// 補間・終点最大値に 10000 を設定する 　[VB.NET]　Call Nmc_TPMax(No, IcNo, 10000) 　[C#]　　MC8000P.Nmc_TPMax(No, IcNo, 10000);
Nmc_HLNumber	ヘリカル回転数を設定する。 **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); **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 **戻り値** 　なし **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);
Nmc_HLValue	ヘリカル演算値を設定する。 **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); **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 **戻り値** 　なし **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);

関数名	機能　及び　内容
Nmc_MRmMode	多目的レジスタモードを設定する。 **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); **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 **戻り値** なし **使用例** // Ｘ軸多目的レジスタモードにＭＲ１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);

Nmc_PIO1Mode	ＰＩＯ信号設定１(nPIO7～0の機能)を設定する。
	
	VC　　void Nmc_PIO1Mode(int No, int IcNo, int axis, long WR6_data);
	VB.NET　Sub　Nmc_PIO1Mode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer)
	C#　　void MC8000P.Nmc_PIO1Mode(int No, int IcNo, AXIS axis, int WR6_data);
	
	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
	
	戻り値
	なし
	
	使用例
	[VC]　　Nmc_PIO1Mode(No, IcNo, AXIS_X, 0x0055); //Ｘ軸ＰＩＯ信号０～３は汎用出力を設定する
	[VB.NET]　Call Nmc_PIO1Mode(No, IcNo, AXIS_X, &H55)
	[C#]　　MC8000P.Nmc_PIO1Mode(No, IcNo, AXIS.X, 0x0055);

Nmc_PIO2Mode	ＰＩＯ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。 **VC**　　void Nmc_PIO2Mode(int No, int IcNo, int axis, long WR6_data); **VB.NET**　Sub　Nmc_PIO2Mode(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer, ByVal WR6_data As Integer) **C#**　　void MC8000P.Nmc_PIO2Mode(int No, int IcNo, AXIS axis, int WR6_data); **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 **戻り値** なし **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_PIO2Mode(No, IcNo, AXIS_X, 0x0070); [VB.NET]　Call Nmc_PIO2Mode(No, IcNo, AXIS_X, &H70) [C#]　　MC8000P.Nmc_PIO2Mode(No, IcNo, AXIS.X, 0x0070);

関数名	機能　及び　内容
Nmc_HMSrch1Mode	自動原点出しモード設定１を設定する。 **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); **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 **戻り値** なし **使用例** Ｘ軸自動原点出しモード設定１に、以下の設定をする。 ・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 [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);
Nmc_HMSrch2Mode	自動原点出しモード設定２を設定する。 **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); **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 **戻り値** なし **使用例** Ｘ軸自動原点出しモード設定２に、以下の設定をする。 ・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：10μsec ・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 [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);

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_FilterMode</td><td>入力信号フィルタモードを設定する。

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);

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

戻り値
なし

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

[VC]　　Nmc_FilterMode(No, IcNo, AXIS_X, 0xAAFF);
[VB.NET]　Call Nmc_FilterMode(No, IcNo, AXIS_X, &HAAFF)
[C#]　　MC8000P.Nmc_FilterMode(No, IcNo, AXIS.X, 0xAAFF);</td></tr>
<tr><td>Nmc_Sync0Mode</td><td>同期動作ＳＹＮＣ０を設定する。

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);

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

戻り値
なし

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

[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);</td></tr>
<tr><td>Nmc_Sync1Mode</td><td>同期動作ＳＹＮＣ１を設定する。

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);

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

戻り値
なし

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

[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);</td></tr>
</table>

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_Sync2Mode</td><td>同期動作ＳＹＮＣ２を設定する。

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);

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

戻り値
なし

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、ＳＹＮＣ３を起動する」を設定する

[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);</td></tr>
<tr><td>Nmc_Sync3Mode</td><td>同期動作ＳＹＮＣ３を設定する。

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);

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

戻り値
なし

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ＳＹＮＣ０を起動する」を設定する

[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);</td></tr>
<tr><td>Nmc_IPMode</td><td>補間モードを設定する。

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);

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

戻り値
なし

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);</td></tr>
</table>

関数名	機能　及び　内容
Nmc_ReadLp	論理位置カウンタを読み出す。 **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); **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。 **戻り値** 　現在の論理位置カウンタの値 **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);
Nmc_ReadEp	実位置カウンタを読み出す。 **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); **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。 **戻り値** 　現在の実位置カウンタの値 **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)
Nmc_ReadSpeed	現在ドライブ速度を読み出す。 **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); **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。 **戻り値** 　現在ドライブ速度の値 **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);

関数名	機能　及び　内容
Nmc_ReadAccDec	現在加／減速度を読み出す。 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。 **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); **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。 **戻り値** 現在加／減速度の値 **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);
Nmc_ReadMR0	多目的レジスタ０を読み出す。 **VC**　long　Nmc_ReadMR0(int No, int IcNo, int axis); **VB.NET**　Function Nmc_ReadMR0(ByVal No As Integer, ByVal IcNo As Integer, ByVal axis As Integer) As Integer **C#**　int　MC8000P.Nmc_ReadMR0(int No, int IcNo, AXIS axis); **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。 **戻り値** 多目的レジスタ０の値 **使用例** [VC]　Data = Nmc_ReadMR0(No, IcNo, AXIS_X);　// Ｘ軸の多目的レジスタ０を読み出す [VB.NET]　Data = Nmc_ReadMR0(No, IcNo, AXIS_X) [C#]　Data = MC8000P.Nmc_ReadMR0(No, IcNo, AXIS.X);
Nmc_ReadMR1	多目的レジスタ１を読み出す。 **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); **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。 **戻り値** 多目的レジスタ１の値 **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);

関数名	機能　及び　内容
Nmc_ReadMR2	多目的レジスタ２を読み出す。 **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); **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。 **戻り値** 多目的レジスタ２の値 **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);
Nmc_ReadMR3	多目的レジスタ３を読み出す。 **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); **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。 **戻り値** 多目的レジスタ３の値 **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);
Nmc_ReadCT	現在タイマー値を読み出す。 **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); **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。 **戻り値** 現在タイマー値の値 **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);

関数名	機能　及び　内容
Nmc_ReadTX	補間・終点最大値を読み出す。 **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); **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。 **戻り値** 補間・終点最大値の値 **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); **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。
Nmc_ReadCHLN	現在ヘリカル回転数を読み出す **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); **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。 **戻り値** 現在ヘリカル回転数の値 **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);
Nmc_ReadHLV	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。 **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); **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。 **戻り値** ヘリカル演算値の値 **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);

関数名	機能　及び　内容
Nmc_ReadWR1	ＷＲ１設定値を読み出す。 **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); **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。 **戻り値** ＷＲ１の値 **使用例** [VC]　Data = Nmc_ReadWR1(No, IcNo, AXIS_Y);　// Ｙ軸のＷＲ１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);
Nmc_ReadWR2	ＷＲ２設定値を読み出す。 **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); **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。 **戻り値** ＷＲ２の値 **使用例** [VC]　Data = Nmc_ReadWR2(No, IcNo, AXIS_Z);　// Ｚ軸のＷＲ２設定値を読み出す [VB.NET]　Data = Nmc_ReadWR2(No, IcNo, AXIS_Z)　'Ｚ軸のＷＲ２設定値を読み出す [C#]　Data = MC8000P. Nmc_ReadWR2(No, IcNo, AXIS.Z);
Nmc_ReadWR3	ＷＲ３設定値を読み出す。 **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_ReadMR3(int No, int IcNo, AXIS axis); **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。 **戻り値** ＷＲ３の値 **使用例** [VC]　Data = Nmc_ReadWR3(No, IcNo, AXIS_U);　// Ｕ軸のＷＲ３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);

関数名	機能　及び　内容

Nmc_ReadMRM	多目的レジスタモード設定を読み出す。
	
	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);
	
	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。
	
	戻り値
	多目的レジスタモードの値
	
	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);

Nmc_ReadP1M	ＰＩＯ信号設定１を読み出す。 **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); **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。 **戻り値** ＰＩＯ信号設定１の値 **使用例** [VC]　Data = Nmc_ReadP1M(No, IcNo, AXIS_Y);　// Ｙ軸のＰＩＯ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);
Nmc_ReadP2M	ＰＩＯ信号設定２・その他設定を読み出す。 **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); **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。 **戻り値** ＰＩＯ信号設定２・その他設定の値 **使用例** // Ｚ軸のＰＩＯ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);

関数名	機能　及び　内容
Nmc_ReadAc	加速度設定値を読み出す。 **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); **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。 **戻り値** 設定した加速度の値 **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);
Nmc_ReadStartSpd	初速度設定値を読み出す。 **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); **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。 **戻り値** 設定した初速度の値 **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);
Nmc_ReadSetSpeed	ドライブ速度設定値を読み出す。 **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); **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。 **戻り値** 設定したドライブ速度の値 **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);

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_ReadPulse</td><td>移動パルス数/終点設定値を読み出す。

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);

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。
戻り値
　設定した移動パルス数/終点設定値の値

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);</td></tr>
<tr><td>Nmc_GetDriveStatus</td><td>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。

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);

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。

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Ｏ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、Ｏ以外を返す。

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″);</td></tr>
</table>

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_GetCNextStatus</td><td>連続補間次データ書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。

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);

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。

戻り値
連続補間次データ書込み可能の場合は、０以外を返す。
連続補間次データ書込み可能ではない場合は、０を返す。

使用例
[VC]　if(Nmc_GetCNextStatus(No, IcNo) != 0)　// 次データ書込み可能の場合
AfxMessageBox("連続補間次データ書込み可能である");
else
AfxMessageBox("連続補間次データ書込み可能ではない");
[VB.NET]　If Nmc_GetCNextStatus(No, IcNo) <> 0 Then
Call MsgBox("連続補間次データ書込み可能である")
Else
Call MsgBox("連続補間次データ書込み可能ではない")
End If
[C#]　if(MC8000P.Nmc_GetCNextStatus(No, IcNo) != 0)
MessageBox.Show("連続補間次データ書込み可能である");
else
MessageBox.Show("連続補間次データ書込み可能ではない");</td></tr>
<tr><td>Nmc_GetSc</td><td>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。

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);

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。

戻り値
現在のプリバッファスタックカウンタの値

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)</td></tr>
</table>

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_WriteRegSetAxis</td><td>指定軸の指定したライトレジスタ(ＷＲ１～ＷＲ３のいずれか)にデータを書き込む。

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);

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。
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

戻り値
なし

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);</td></tr>
<tr><td>Nmc_ReadRegSetAxis</td><td>指定軸の指定したリードレジスタ(ＲＲ２、ＲＲ３のどちらか)のデータを読み出す。

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);

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。
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。

戻り値
指定軸の指定したリードレジスタのデータ

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);

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。</td></tr>
</table>

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_WriteData</td><td>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)

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);

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

戻り値
なし

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);</td></tr>
<tr><td>Nmc_ReadData</td><td>データ読み出し命令を実行し、データを読み出す。

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);

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。

戻り値
読み出したデータ

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);</td></tr>
</table>

関数名	機能　及び　内容

Nmc_2BPExecMC8500P	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。
	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。
	
	VC　DWORD　Nmc_2BPExecMC8500P(int No, int IcNo, DATA_2BP* pData2Bp, int DataCnt, int IpAxis);
	
	VB.NET　Function Nmc_2BPExecMC8500P(ByVal No As Integer, ByVal IcNo As Integer, ByVal pData2Bp As DATA_2BP(), ByVal DataCnt As Integer, ByVal IpAxis As Integer) As Integer
	
	C#　Nmc_Status MC8000P.Nmc_2BPExecMC8500P(int No, int IcNo, DATA_2BP[] pData2Bp, int DataCnt, int IpAxis);
	
	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
	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「5.1.3　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。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。
	
	戻り値
	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。
	
	■正常終了
	BP_END　BP補間処理正常終了
	
	■エラーコード
	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
	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
	
	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。
	
	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_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)
	Ret = Nmc_2BPExecMC8500P(No, IcNo, Data2Bp, 2, 0x3);//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
	
	[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)

        Ret = Nmc_2BPExecMC8500P(No, IcNo, Data2Bp, 2, &H3) '２軸ＢＰ補間実行。データ数数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 = MC8000P.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。

関数名	機能　及び　内容

Nmc_3BPExecMC8500P	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。
	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。
	
	VC　DWORD　Nmc_3BPExecMC8500P(int No, int IcNo, DATA_3BP* pData3Bp, int DataCnt, int IpAxis);
	
	VB.NET　Function Nmc_3BPExecMC8500P(ByVal No As Integer, ByVal IcNo As Integer, ByVal pData3Bp As DATA_3BP(), ByVal DataCnt As Integer, ByVal IpAxis As Integer) As Integer
	
	C#　Nmc_Status MC8000P.Nmc_3BPExecMC8500P(int No, int IcNo, DATA_3BP[] pData3Bp, int DataCnt, int IpAxis);
	
	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
	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「5.1.3　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。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。
	
	戻り値
	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。
	
	■正常終了
	BP_END　BP補間処理正常終了
	
	■エラーコード
	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
	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
	
	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。
	
	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_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 = Nmc_3BPExecMC8500P(No, IcNo, Data3Bp, 2, 0x7); //３軸ＢＰ補間実行。データ数2, X,Y,Z軸
	if(Ret == BP_END)　AfxMessageBox("正常終了");　//戻り値正常
	
	[VB.NET]
	'３軸ＢＰ補間データ
	'補間データ設定
	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 = &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[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)

     Ret = MC8000P.Nmc_3BPExecMC8500P(No, IcNo, Data3Bp, 2, 0x7); // ３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。

関数名	機能　及び　内容

Nmc_4BPExecMC8500P	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。
	
	VC　DWORD　Nmc_4BPExecMC8500P(int No, int IcNo, DATA_4BP* pData4Bp, int DataCnt, int IpAxis);
	
	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
	
	C#　Nmc_Status MC8000P.Nmc_4BPExecMC8500P(int No, int IcNo, DATA_4BP[] pData4Bp, int DataCnt, int IpAxis);
	
	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
	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。
	
	戻り値
	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。
	
	■正常終了
	BP_END　BP補間処理正常終了
	
	■エラーコード
	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
	
	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。
	
	使用例
	[VC]
	// 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
	
	[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(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。

関数名	機能　及び　内容
Nmc_2BPExecMC8500P_BG	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。 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。 **VC**　DWORD　Nmc_2BPExecMC8500P_BG(HWND User_hWnd, int No, int IcNo, DATA_2BP* pData2Bp, int DataCnt, int IpAxis); **VB.NET**　Function Nmc_2BPExecMC8500P_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByVal pData2Bp As DATA_2BP(), ByVal DataCnt As Integer, ByVal IpAxis As Integer) As Integer **C#**　Nmc_Status MC8000P.Nmc_2BPExecMC8500P_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_2BP[] pData2Bp, int DataCnt, int IpAxis,); **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「5.1.3　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「5.1.3　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。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。 **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 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「5.1.3　補足説明」(1)参照。 ■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 ■エラーコード(補間開始前のエラー) 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_THREAD_ERR　スレッドを起動できなかった 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 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。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード(補間開始後のエラー) 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） 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 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。 **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_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)

        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ください。

関数名	機能　及び　内容
Nmc_3BPExecMC8500P_BG	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。 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。

VC　　DWORD　Nmc_3BPExecMC8500P_BG(HWND User_hWnd, int No, int IcNo, DATA_3BP* pData3Bp, int DataCnt, int IpAxis);

VB.NET　Function Nmc_3BPExecMC8500P_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByVal pData3Bp As DATA_3BP(), ByVal DataCnt As Integer, ByVal IpAxis As Integer) As Integer

C#　　Nmc_Status MC8000P.Nmc_3BPExecMC8500P_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_3BP[] pData3Bp, int DataCnt, int IpAxis,);

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「5.1.3　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「5.1.3　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。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。

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 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「5.1.3　補足説明」(1)参照。

■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

■エラーコード(補間開始前のエラー)

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_THREAD_ERR	スレッドを起動できなかった
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

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。

■正常終了

BP_END	BP補間処理正常終了

■エラーコード(補間開始後のエラー)

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）
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

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。

使用例

```
[VC]
    {
      // ３軸ＢＰ補間データ    BP1P,     BP1M,    BP2P,    BP2M,    BP3P,    BP3M
      DATA_3BP Data3Bp[2] = {{0xFF30, 0x0000, 0x0000, 0x84FF, 0x0000, 0xAC35},
                             {0xAC35, 0x0000, 0xC000, 0x36E7, 0xC000, 0x3F3F}};

      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 = 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ください。

関数名	機能　及び　内容
Nmc_4BPExecMC8500P_BG	指定した補間データで４軸ビットパターン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_BP_ENDメッセージを送信し、終了ステータスを渡します。 **VC**　DWORD　Nmc_4BPExecMC8500P_BG(HWND User_hWnd, int No, int IcNo, DATA_4BP* pData4Bp, int DataCnt, int IpAxis); **VB.NET**　Function Nmc_4BPExecMC8500P_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByVal pData4Bp As DATA_4BP(), ByVal DataCnt As Integer, ByVal IpAxis As Integer) As Integer **C#**　Nmc_Status MC8000P.Nmc_4BPExecMC8500P_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_4BP[] pData4Bp, int DataCnt, int IpAxis,); **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「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。 **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 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「5.1.3　補足説明」(1)参照。 ■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 ■エラーコード(補間開始前のエラー) 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_THREAD_ERR　スレッドを起動できなかった 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 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。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード(補間開始後のエラー) 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） 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 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。 **使用例** [VC] { // 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)

      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ください。

関数名	機能　及び　内容

Nmc_2CIPExecMC8500P	指定した補間データで２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。
	
	VC　　DWORD　Nmc_2CIPExecMC8500P(int No, int IcNo, DATA_2CIP_MC8500P* pData2Cip,
	int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE);
	
	VB.NET　Function Nmc_2CIPExecMC8500P(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
	
	C#　　Nmc_Status MC8000P.Nmc_2CIPExecMC8500P(int No, int IcNo, DATA_2CIP_MC8500P[] pData2Cip,
	int DataCnt,int IpAxis, bool SpdChgFlg);
	
	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
	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。
	
	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「5.1.3　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_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
	CIP_STOP　　連続補間が途中で停止した
	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
	
	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]
	// 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(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。

関数名	機能　及び　内容
Nmc_3CIPExecMC8500P	指定した補間データで３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。

VC　　DWORD　Nmc_3CIPExecMC8500P(int No, int IcNo, DATA_3CIP_MC8500P* pData3Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE);

VB.NET　Function Nmc_3CIPExecMC8500P(ByVal No As Integer, ByVal IcNo As Integer, ByVal pData3Cip As DATA_3CIP_MC8500P(), ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer) As Integer

C#　　Nmc_Status MC8000P.Nmc_3CIPExecMC8500P(int No, IcNo, DATA_3CIP_MC8500P[] pData3Cip, int DataCnt,int IpAxis, bool SpdChgFlg);

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
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_MC8500Pのアドレス)。 DATA_3CIP_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_3CIP_MC8500Pについては「5.1.3　補足説明」(3)参照。
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。
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　：続ける、false　：続けない **変更する選択時**　：DATA_3CIP_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_3CIP_MC8500PのSpeedに設定した値を参照しません。

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「5.1.3　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_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
CIP_STOP	連続補間が途中で停止した
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

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[0].Speed = 0;
```

```
        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ください。

関数名	機能　及び　内容
Nmc_4CIPExecMC8500P	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。 **VC**　DWORD　Nmc_4CIPExecMC8500P(int No, int IcNo, DATA_4CIP_MC8500P* pData4Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE); **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 **C#**　Nmc_Status MC8000P.Nmc_4CIPExecMC8500P(int No, IcNo, DATA_4CIP_MC8500P[] pData4Cip, int DataCnt,int IpAxis, bool SpdChgFlg); **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 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　：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。 **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「5.1.3　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_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 CIP_STOP　連続補間が途中で停止した 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 **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[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	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。 **VC**　DWORD　Nmc_2CIPExecMC8500P_BG(HWND User_hWnd, int No, int IcNo, DATA_2CIP_MC8500P* pData2Cip, int DataCnt,int IpAxis, BOOL SpdChgFlg = FALSE); **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 **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); **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「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。 **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「5.1.3　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 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_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 **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]

```
 {
  // 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 '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ください。

関数名	機能　及び　内容
Nmc_3CIPExecMC8500P_BG	指定した補間データで３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。 **VC**　DWORD　Nmc_3CIPExecMC8500P_BG(HWND User_hWnd, int No, int IcNo, DATA_3CIP_MC8500P* pData3Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE); **VB.NET**　Function Nmc_3CIPExecMC8500P_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByVal pData3Cip As DATA_3CIP_MC8500P(), ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer) As Integer **C#**　Nmc_Status MC8000P.Nmc_3CIPExecMC8500P_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_3CIP_MC8500P[] pData3Cip, int IpAxis, bool SpdChgFlg); **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「5.1.3　補足説明」(7)参照 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_MC8500Pのアドレス)。 DATA_3CIP_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_3CIP_MC8500Pについては「5.1.3　補足説明」(3)参照。 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。 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_3CIP_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_3CIP_MC8500PのSpeedに設定した値を参照しません。 **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「5.1.3　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 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_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

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_3CIPExecMC8500P_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ください。

関数名	機能　及び　内容
Nmc_4CIPExecMC8500P_BG	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。 **VC**　　DWORD　Nmc_4CIPExecMC8500P_BG(HWND User_hWnd, int No, int IcNo, DATA_4CIP_MC8500P* pData4Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE); **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 **C#**　　Nmc_Status MC8000P.Nmc_4CIPExecMC8500P_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_4CIP_MC8500P[] pData4Cip, int IpAxis, bool SpdChgFlg); **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「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。 **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「5.1.3　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 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_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

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_MC850OP 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_4CIPExecMC850OP_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_MC850OP
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_4CIPExecMC850OP_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
                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_4CIPExecMC8500P_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。

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_IPStop</td><td>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。

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

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);

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。

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

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);</td></tr>
<tr><td>Nmc_IPGetMsgNo</td><td>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

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)

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。

戻り値
なし

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)</td></tr>
</table>

関数名	機能　及び　内容

Nmc_HLValueExec	指定した補間データでヘリカル演算を実行する。
	
	VC　DWORD　Nmc_HLValueExec(int No, int IcNo, DATA_HL* pDataHl);
	VB.NET　Function　Nmc_HLValueExec(ByVal No As Integer, ByVal IcNo As Integer,
	ByRef pDataHl As DATA_HL) As Integer
	C#　Nmc_Status　Nmc_HLValueExec(int No, int IcNo, ref DATA_HL pDataHl);
	
	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。
	pDataHl　ヘリカル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_HLのアドレス)。
	DATA_HLに実行するヘリカル演算のデータをセットし、アドレスに指定する。
	詳細は「5.1.3　補足説明」(3)参照。
	DirectionにはＣＷ円弧補間(HL_DIR_CW)、ＣＣＷ円弧補間(HL_DIR_CCW)のどちらかを指定する。
	
	戻り値
	演算処理が正常終了した場合は HL_VALUE_END が返ります。
	演算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。
	
	■正常終了
	HL_VALUE_END　ヘリカル演算正常終了
	
	■エラーコード
	HL_PARAM_ERR　引数の値が正しくない
	HL_NOT_OPEN_ERR　指定したボードがオープンされていない
	HL_FUNC_ERR　使用できない関数を使用
	HL_OTHER_ERR　その他のエラー
	
	使用例
	[VC]
	DATA_IPMODE IpMode;　// 補間モード
	
	// 補間モード設定
	IpMode.Cxiv = FALSE;　// 補間軸入れ替え無し
	IpMode.Lmdf = TRUE;　// 短軸パルス均一化有効
	IpMode.Vcnst = 3;　// 2軸高精度線速一定
	
	DATA_HL DataHL;　// ヘリカル補間データ
	
	// 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;　// 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.HlNum = 1;　// ヘリカル回転数
	DataHL.HlValue = 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
	Ret = Nmc_HLValueExec(0, 0, &DataHL);
	if(Ret == HL_VALUE_END)　AfxMessageBox("演算終了");　// ヘリカル演算正常終了
	
	// ヘリカル演算値取得
	Value = DataHL.HlValue;　// 演算終了後、演算結果がHlValueに格納される
	
	[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い
	IpMode.Lmdf = True　'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                 '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.HlNum = 1                 'ヘリカル回転数
    DataHL.HlValue = 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.HlValue         '演算終了後、演算結果がHlValue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.HlNum = 1;                    // ヘリカル回転数
    DataHL.HlValue = 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.HlValue;            // 演算終了後、演算結果がHlValue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下さい。
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。

関数名	機能　及び　内容
Nmc_HLExec	指定した補間データでヘリカル補間を実行する。

VC　　DWORD　　Nmc_HLExec(int No, int IcNo, DATA_HL* pDataHl);
VB.NET　Function Nmc_HLExec(ByVal No As Integer, ByVal IcNo As Integer,
ByRef pDataHl As DATA_HL) As Integer
C#　　Nmc_Status　Nmc_HLExec(int No, int IcNo, ref DATA_HL pDataHl);

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。
pDataHl　　ヘリカル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_HLのアドレス)。
DATA_HLに実行するヘリカル演算のデータをセットし、アドレスに指定する。
詳細は「5.1.3　補足説明」(3)参照。

ヘリカル補間データの構造体の変数 ヘリカル演算値HLValue の値を0に設定すると、ドライバ内でヘリカル演算を行い、演算で算出された値を使用してヘリカル補間を行います。ヘリカル演算値HLValueは演算により算出された演算値に書き換えられます。
ヘリカル演算値を0以上に設定するとドライバ内ではヘリカル演算を行わず、設定した値を使用してヘリカル補間を行います。

戻り値

ヘリカル補間コマンドが正常に書き込まれた場合は HL_START が返ります。
ヘリカル補間が開始する前にエラーが発生した場合は、下記のヘリカル補間開始前のエラーコードが返ります。

■正常開始
HL_START　　ヘリカル補間開始

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

使用例

[VC]

```
DATA_IPMODE IpMode;                  // 補間モード

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

DATA_HL DataHL;                      // ヘリカル補間データ

// 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;                    // 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.HlNum = 1;                    // ヘリカル回転数
DataHL.HlValue = 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

// ヘリカル補間実行
Ret = Nmc_HLExec(0, 0, &DataHL);
if(Ret == HL_START)   AfxMessageBox("ヘリカル補間開始");      // ヘリカル補間開始

// ヘリカル演算値取得
Value = DataHL.HlValue;              // 演算結果がHlValueに格納される
```

[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い
    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.HlNum = 1                'ヘリカル回転数
    DataHL.HlValue = 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.HlValue          '演算結果がHlValue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.HlNum = 1;                    // ヘリカル回転数
   DataHL.HlValue = 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.HlValue;            // 演算終了後、演算結果がHlValue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下さい。
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 // ＷＲ０
   #define MCX_WR1        0x0001 // ＷＲ１
   #define MCX_WR2        0x0002 // ＷＲ２
   #define MCX_WR3        0x0003 // ＷＲ３
   #define MCX_WR4        0x0004 // ＷＲ４
   #define MCX_WR5        0x0005 // ＷＲ５
   #define MCX_WR6        0x0006 // ＷＲ６
   #define MCX_WR7        0x0007 // ＷＲ７

   #define MCX_RR0        0x0000 // ＲＲ０
   #define MCX_RR1        0x0001 // ＲＲ１
   #define MCX_RR2        0x0002 // ＲＲ２
   #define MCX_RR3        0x0003 // ＲＲ３
   #define MCX_RR4        0x0004 // ＲＲ４
   #define MCX_RR5        0x0005 // ＲＲ５
   #define MCX_RR6        0x0006 // ＲＲ６
   #define MCX_RR7        0x0007 // ＲＲ７
```

[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
MC8541P / MC8541Pe	0xA808
MC8581P / MC8581Pe	0xA809
MC8543PeL	0xB001

[VC]

```
#define ID_MC8541P      0xA808        // MC8541P及びMC8541Pe
#define ID_MC8581P      0xA809        // MC8581P及びMC8581Pe
#define ID_MC8543PEL    0xB001        // MC8543PeL
```

[C#]

```
public enum Dev_ID : ushort
{
    MC8541P             =0xA808,      // MC8541P及びMC8541Pe
    MC8581P             =0xA809       // MC8581P及びMC8581Pe
    MC8543PEL           =0x B001      // MC8543PeL
}
```

（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_BPIP2        0x0066     // 2軸ビットパターン補間ドライブ
#define MC8500P_CMD_BPIP3        0x0067     // 3軸ビットパターン補間ドライブ
#define MC8500P_CMD_BPIP4        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)// 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
#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(速度が速く次ﾃﾞ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ＢＰ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(速度が速く次ﾃﾞ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) ＢＰ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ＢＰ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ＢＰ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

①１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);
```

②２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);
```

③３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。
①ＶＣ

```
// 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;
```

②ＶＢ．ＮＥＴ

```
’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（１～２０００００）
         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
```

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
}

 // 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:ＣＷ円弧補間, HL_DIR_CCW：ＣＣＷ円弧補間のいずれかをセットする）
      public int     Speed;      // 速度（１～２０００００）
      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　（０～６５５３５）
      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。（ＶＣ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実例のデータ)
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
MC8543PeL	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。

①ＩＣを１つ搭載しているボードの場合は０を指定します。

②ＩＣ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
MC8543PeL	1	0

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

① Nmc_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。

③ １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。

④ Nmc_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。

⑤ 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。

⑥ 割込みを行う場合は、割込みユーザー関数（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にて割込みを処理する方法

① Nmc_Open関数にて割込みを使用する設定にします。

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

② Nmc_SetEvent関数にて割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の停止時割込み発生（全軸）
```

③ 割込みが発生すると、Nmc_SetEvent関数で設定した割込みユーザー関数が呼び出されます。
割込みユーザー関数では、割込み要因を確認します。RR1の割込み要因は、Nmc_ReadEvent関数にて取得します。

■割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);
```

■ＶＢ．ＮＥＴにて割込みを処理する方法

ＶＣと同じ手順で処理をします。

■Ｃ＃にて割込みを処理する方法（Ｃ＃のみ）

①　MC8000P.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４０００００００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。

■ＢＰ補間（ビットパターン補間）について

ＢＰ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、ＢＰ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ＢＰ補間関数を使用してＢＰ補間の処理を行うこともできます。但し、ＢＰ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

ＢＰ補間関数を使用する場合の注意事項：

ＢＰ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、ＢＰ補間関数は使用せず、MCX514取説明書を参考にしてアプリケーションでＢＰ補間の処理を作成してください。また、ＢＰ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、ＷＲ６，ＷＲ７にデータを書き込む処理、ＲＲ６，ＲＲ７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		

◆ＷＲ６，ＷＲ７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

◆ＲＲ６，ＲＲ７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

ＷＲ１～ＷＲ３書き込み、ＲＲ２～ＲＲ３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。

◆ＷＲ１～ＷＲ３書き込み

Nmc_WriteReg1　Nmc_WriteReg2　Nmc_WriteReg3　Nmc_WriteRegSetAxis

◆ＲＲ２～ＲＲ３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

ＷＲ１～ＷＲ３書き込み、ＲＲ２～ＲＲ３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、ＷＲ１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（ＩＣ－Ａ）
②  Nmc_OutPort(No, MCX_WR1,  Data);          // ＷＲ１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,  0x011F);// Ｘ軸に切り替える
④  Nmc_WriteReg(No, IcNo, MCX_WR1,  Data);  // ＷＲ１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ＷＲ１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 );        // ＷＲ６書き込み（ＩＣ－Ａ）
②  Nmc_OutPort( No, MCX_WR0, 0x0105 );      // ＷＲ６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);           // ＷＲ６書き込み
④  Nmc_Command(No, IcNo, AXIS_X, 0x05);     // ＷＲ６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、ＷＲ６，ＷＲ７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ＲＲ６，ＲＲ７に読み出す（ＩＣ－Ａ）
②  d6 = Nmc_InPort( No, MCX_RR6 );     // ＲＲ６を読み出す（ＩＣ－Ａ）
③  d7 = Nmc_InPort( No, MCX_RR7 );     // ＲＲ７を読み出す（ＩＣ－Ａ）
```

この場合、①と②の間、②と③の間に、ＲＲ６，ＲＲ７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に入力信号フィルタモード設定コマンド（25h）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：512μsec

各信号のフィルタの有効/無効はボードによって異なります。

A)　MC8541P/MC8581Pの場合

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

B)　MC8543PeLの場合

信号名	有効 / 無効
EMG	有効
nLMTP, nLMTM	有効
nSTOP0, nSTOP1	有効
nINPOS, nALARM	有効
nPIO0	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。入力信号フィルタ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は入力信号フィルタ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）汎用入出力信号初期設定

汎用入出力信号設定はボードによって異なります。

A) MC8541P/MC8581Pの場合

■ nPIOm信号の機能設定

パソコン起動時の初期設定において、MCX514にPIO信号設定１コマンド（21h）を発行して各入力信号に対して、以下のように機能設定を行います。

信号名	汎用入力 / 汎用出力
nPIO7	入力
nPIO6	入力
nPIO5	入力
nPIO4	入力
nPIO3	出力
nPIO2	出力
nPIO1	出力
nPIO0	出力

■ 汎用出力

機能設定で汎用出力に設定した信号に対して、以下のように設定を行います。

信号名	Hi / Low
nPIO3	Lowレベル
nPIO2	Lowレベル
nPIO1	Lowレベル
nPIO0	Lowレベル

アプリケーション上で、これらのPIO信号設定を変更する場合には、MCX514の取扱説明書7.3.2節を参照してください。PIO信号設定１コマンド（21h）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_PIO1ModeはPIO信号設定１コマンド（21h）を実行しています。

B) MC8543PeLの場合

■ nPIOm信号の機能設定

パソコン起動時の初期設定において、MCX514にPIO信号設定１コマンド（21h）を発行して各入力信号に対して、以下のように機能設定を行います。

信号名	汎用入力 / 汎用出力
nPIO7	出力
nPIO6	出力
nPIO5	入力
nPIO4	入力
nPIO3	出力
nPIO2	出力
nPIO1	出力
nPIO0	入力

■ 汎用出力

機能設定で汎用出力に設定した信号に対して、以下のように設定を行います。

信号名	Hi / Low
nPIO7	Hiレベル
nPIO6	Lowレベル
nPIO3	Lowレベル
nPIO2	Lowレベル
nPIO1	Lowレベル

アプリケーション上で、これらのPIO信号設定を変更する場合には、MCX514の取扱説明書7.3.2節を参照してください。PIO信号設定１コマンド（21h）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_PIO1ModeはPIO信号設定１コマンド（21h）を実行しています。

注意：
①コマンドリセット(00FFh)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。

（３）ハードリミット初期設定

パソコン起動時の初期設定において、ハードリミット入力信号に対して、以下のように設定を行います。

信号名	有効 / 無効
HLM-E	有効

アプリケーション上で、ハードリミット入力信号の設定を変更する場合には、MCX514の取扱説明書6.6節を参照してください。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。

（例１）
```
  Nmc_WriteReg2(0, 0, AXIS_ALL, 0x0800);  // ＩＣ－Ａの設定
```

注意：
①コマンドリセット(00FFh)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。

（３）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。

（４）割り込みサポートについて

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

（５）割り込みクリアについて

①　ＲＲ１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、MC8543PeL)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(MC8541P_MC8581P)フォルダ(Tool¥ MCX514 Board(MC8543PeL)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

ご使用になるボードにより、実行プログラムが異なります。
MC8541P/MC8581Pをご使用の場合、実行プログラムはMCX514.exeです。
MC8543PeLをご使用の場合、実行プログラムはMCX514_PeL.exeです。
MCX514.exeはTool¥ MCX514 Board(MC8541P_MC8581P)フォルダに、MCX514_PeL.exeはTool¥ MCX514 Board(MC8543PeL)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

① MCX514を１つ搭載しているボード(MC8541P, MC8541Pe)のMCX514
② MCX514を２つ搭載しているボード(MC8581P, MC8581Pe)のMCX514

●MCX514_PeL.exe

① MCX514を１つ搭載しているボード(MC8543PeL)のMCX514

■実行について

異なるボードに対して複数のMCX514.exe(MCX514_PeL.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(MCX514_PeL.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。

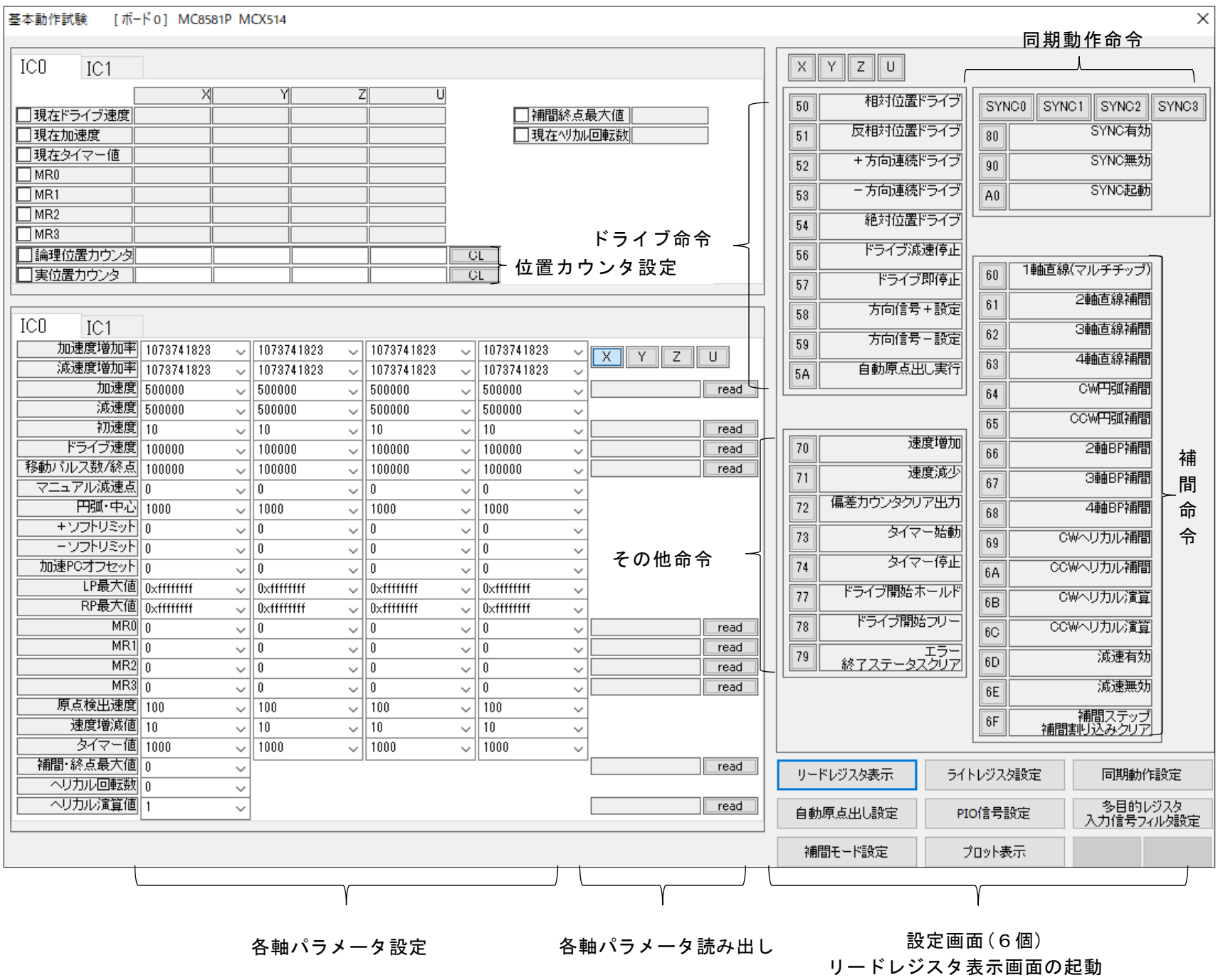
5.3.2　機能概要

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。

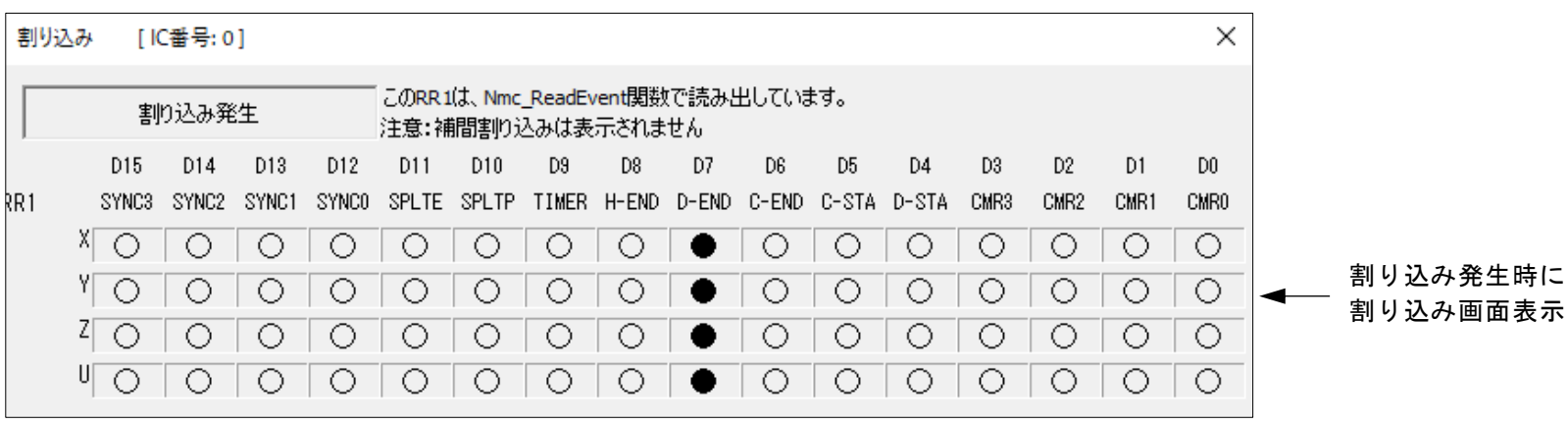
メイン画面では、各軸パラメータ設定および読み出し、ドライブ命令等の命令実行、現在位置・現在速度の表示、割り込み画面表示等を行います。また、モード設定画面(ライトレジスタ・同期動作・自動原点出し・ＰＩＯ・多目的レジスタ・入力信号フィルタ・補間モード設定画面)やリードレジスタ画面を開き、モード設定、リードレジスタ参照等を行う事ができます。

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。

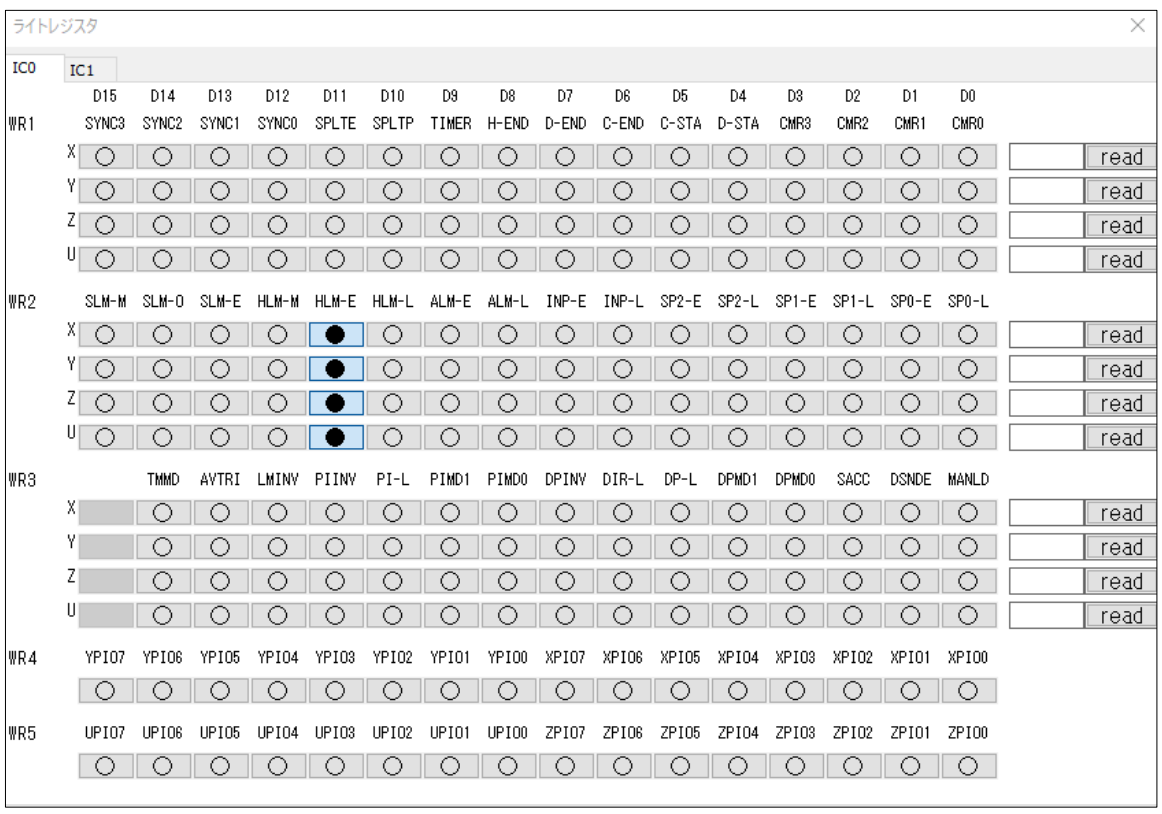


ドライブ中の現在位置や現在ドライブ速度などの表示は、項目の左隣の□にチェックを入れると周期的に値を読み出して表示します。
補間ドライブ中の現在ドライブ速度は、主軸の演算パルス速度を読み出して表示します。



5.3.4　ライトレジスタ設定画面

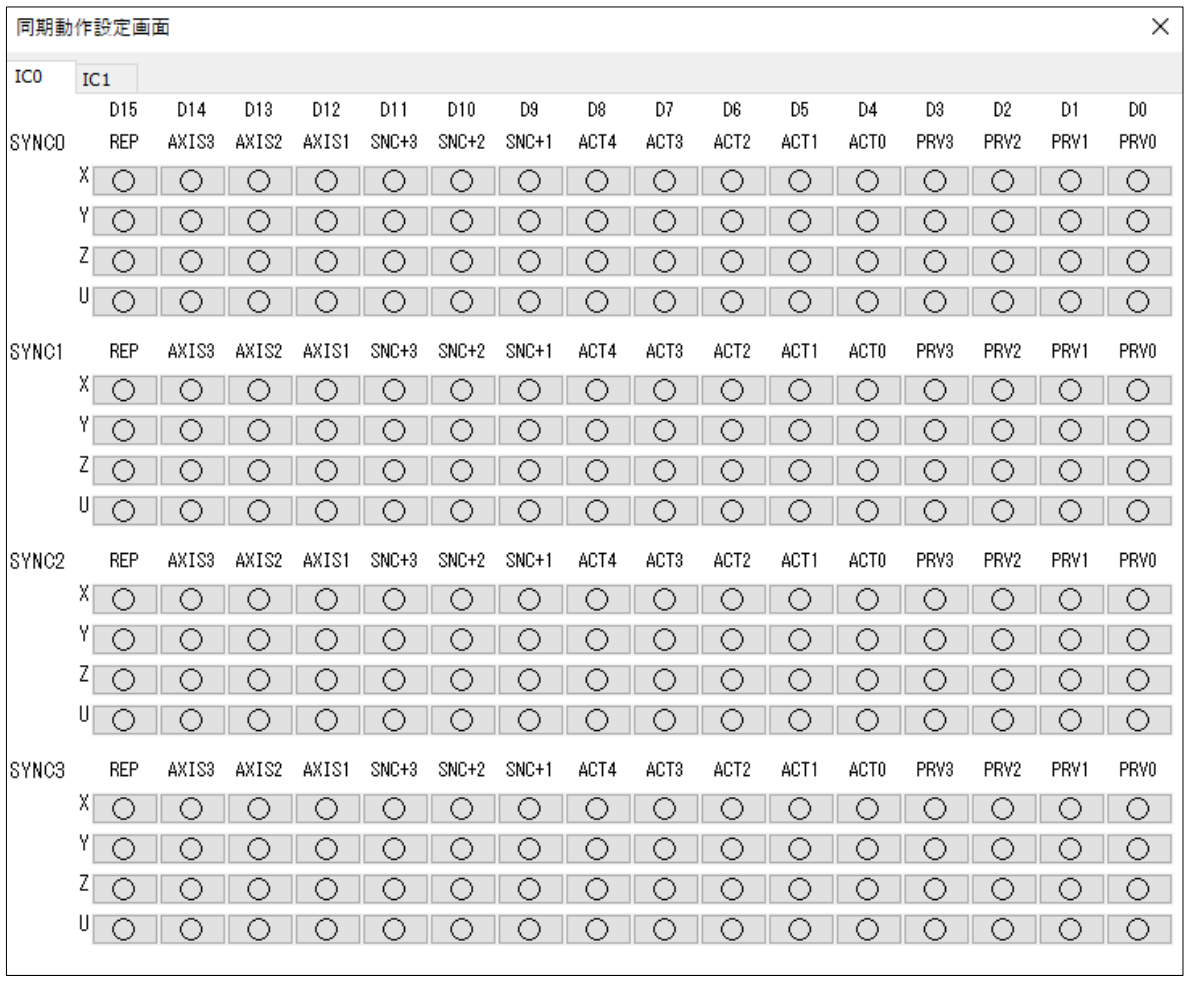
ライトレジスタ設定画面では、MCX514のWR1～WR5（モードレジスタ、アウトプットレジスタ）を設定、WR1～WR3（モードレジスタ）の読み出しをします。



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、MC8543PeLでは使用できるPIO信号が限定されています。ハードウエ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

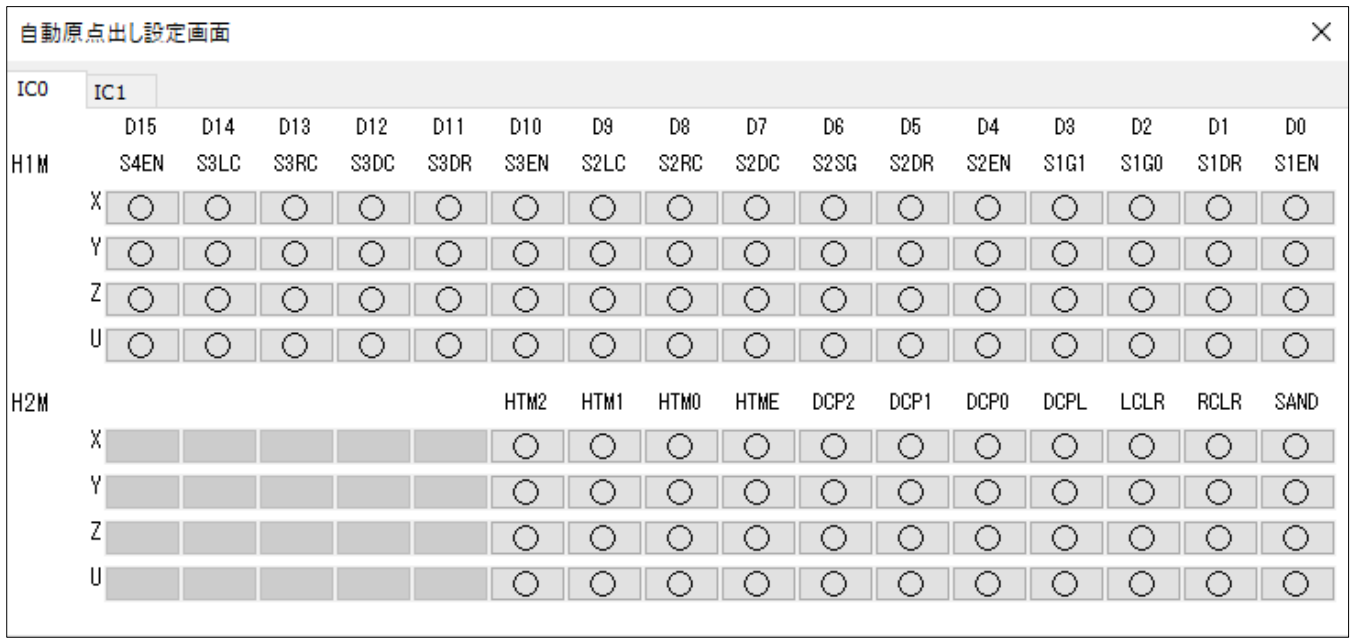
同期動作設定画面では、MCX514の同期動作モードレジスタ（SYNC0～SYNC3）を設定します。



MC8541P/MC8541Pe、MC8581P/MC8581Pe、MC8543PeLでは使用できる同期動作の種類が限定されています。
ハードウエ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

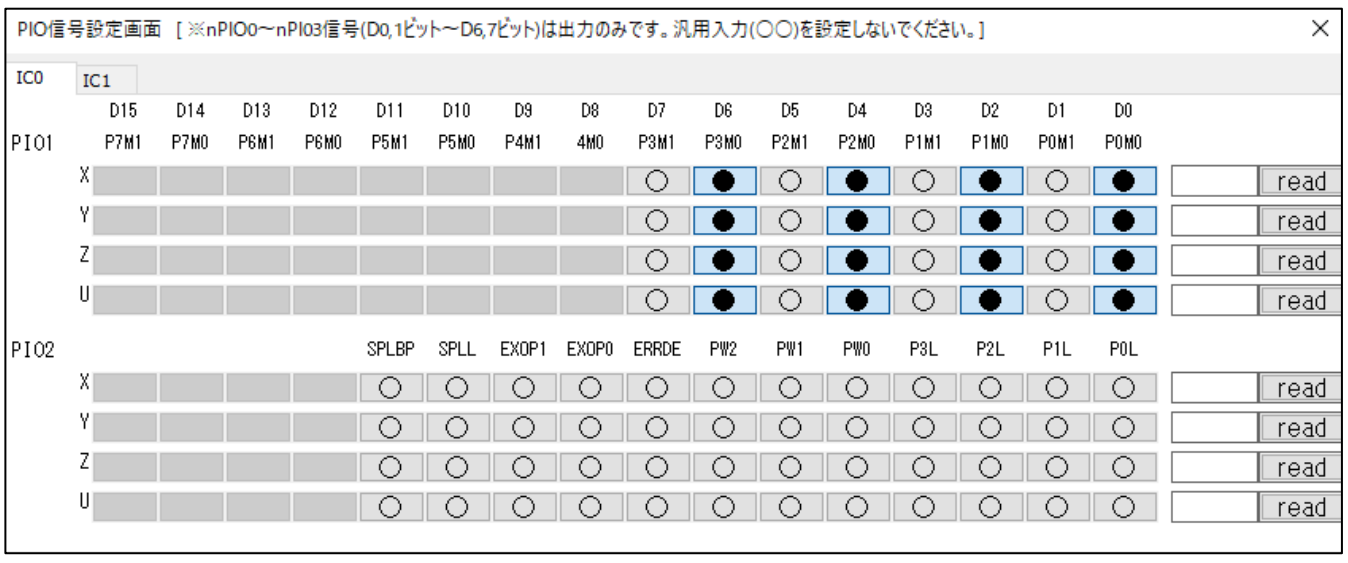
自動原点出し設定画面では、MCX514の自動原点出しレジスタ（H1M,H2M)を設定します。



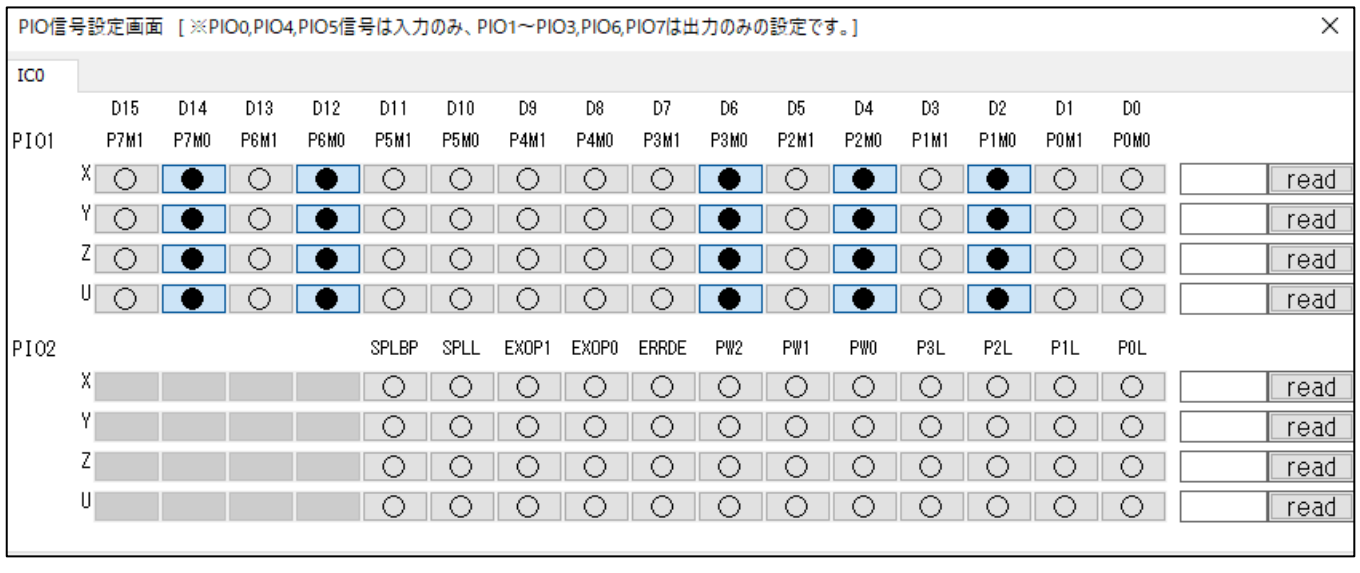
5.3.7　PIO信号設定画面

PIO信号設定画面では、MCX514のPIO（PIO1,PIO2）を設定、読み出しをします。

■　MCX514.exe(MC8541P,MC8581P)

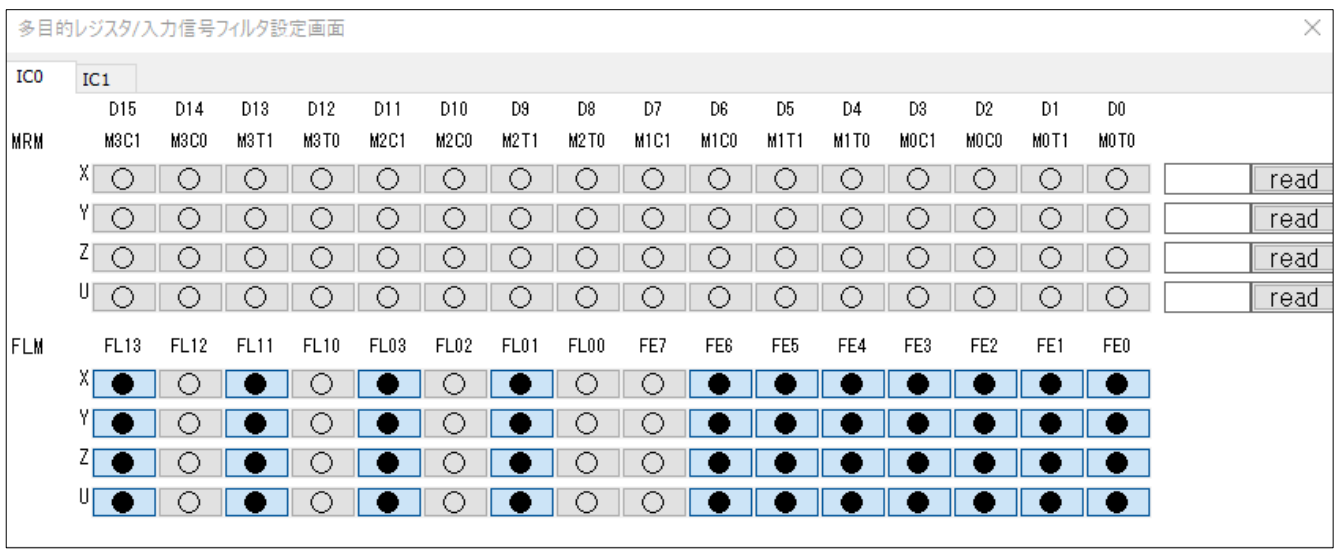


■　MCX514_PeL.exe(MC8543PeL)



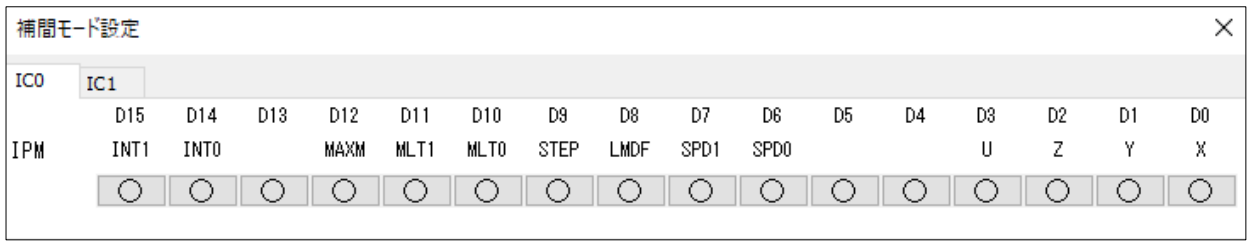
5.3.8　多目的レジスタ/入力信号フィルタ設定画面

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、ＰＩＯ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。