USB/LAN 対応　直線／円弧補間機能付 モーションコントロールユニット

MR540/MR580

取扱説明書・ソフトウェア編

2013. 4. 17　　初　版
2014. 3. 20　　第 2 版
2015. 6. 01　　第 3 版
2016. 7. 12　　第 3.1 版
2022. 3. 02　　第 4 版

NOVA　electronics　　株式会社 ノヴァエレクトロニクス

■改訂履歴

版　数	改訂年月日	改訂内容
初　版	2013年4月17日	新規作成
第2版	2014年3月20日	2.4 節： DHCP サーバのない環境での PC･IP アドレス設定例の追加 2.5 節： Windows OS･64bit 版対応の設定など追加 2.6 節： Windows OS･64bit 版対応の設定など追加 3 章 ： 機能追加の説明を追加 ・デバイスドライバを、Windows OS･32bit 用と 64bit 用に分割 ・API の VB、VB.NET 対応
第3版	2015年6月01日	1.1 節： 対応 OS Windows 8,1 を追加 1.2 節： 対応言語　Microsoft Visual C++ 2013、 Microsoft Visual Basic 2013 を追加 2.3 節： USB ドライバの削除に Windows 8.1 を追加 2.4 節： DHCP サーバのない環境での接続方法 3 章 ： Microsoft Visual C++ 2013、 Microsoft Visual Basic 2013 を追加
第3.1版	2015年7月12日	誤字脱字の修正
第4版	2022年2月25日	対応 OS Windows10 用第4版を作成 第3．1版から下記の内容を Windows10 用に内容を変更 2 章 ： インストール 3 章 ： プログラミング 3.4.10 節　データ書き込み関数の引数の型の変更

■本書で使用する用語

本製品	MR540 または MR580 を示します。
ユニット	MR540 または MR580 を示します。
本取扱説明書	MR540/MR580 取扱説明書・ソフトウェア編
ハードウェア取扱説明書	MR540/MR580 取扱説明書・ハードウェア編
IC	IC とは、MCX314AL を示します。
ドライブ	パルス列入力のサーボモータ、あるいはステッピングモータのドライバ（駆動装置）に対し、モータを回転させるためのパルスを出力する動作。
アクティブ	ある信号において、その信号の持つ機能が有効な状態であること。
PC	本製品を制御するパソコンを示します。
n 信号名	‘n’は、X,Y,Z,U 軸を表します。（例：nLMT+, nLMT－など）
連続補間ドライブ	2 軸直線補間、円弧補間（2 軸）、3 軸直線補間、2 軸 BP 補間、3 軸 BP 補間を連続して補間ドライブを行う場合の総称です。

はじめに

このたびは、本製品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

本製品のモータコントロール IC は、MCX314AL を使用しています。モータ制御の基本機能はすべて MCX314AL に依存していますので、機能動作の詳細については「MCX314As/AL 取扱説明書」を併せてご参照ください。
また、本製品のハードウェア仕様に関しては、「MR540/MR580 取扱説明書・ハードウェア編」 をご参照ください。

■ LAN 通信ご使用の注意

LAN 通信を使用する場合、本製品の IP アドレスは、DHCP サーバのある環境で自動取得されますが、手動での設定はできません。DHCP サーバのない環境では、固定の IP アドレスが設定されます。

― 目 次 ―

1. 概　要

本製品を制御するには、弊社より提供の“MR540/MR580 用ソフトウェア”を用いてお客様専用のアプリケーションを作成する必要があります。“MR540/MR580 用ソフトウェア”には、Windows 用デバイスドライバや API 関数（アプリケーション用制御関数）、サンプルプログラムが用意されており、Windows 用のアプリケーションが容易に作成できます。
MR540/MR580 デバイスドライバは、本製品の専用デバイスドライバです。

1.1 対応 OS

MR540/MR580 デバイスドライバは、Windows 10 (32ビット版、64 ビット版)に対応しています。

1.2 対応言語

MR540/MR580 デバイスドライバは、次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 を呼び出す事により、本製品を制御できます。

対応言語
Microsoft Visual C++
Microsoft Visual Basic

1.3 複数台制御する場合の設定

MR540/MR580 デバイスドライバは、本製品を同時に16台まで認識します。
1 台の PC で、複数台の本製品を制御する場合は、本製品を個別に認識するためにロータリスイッチでユニット番号を設定します。ユニット番号の範囲は0～F(15)で、それぞれの製品において重複しないように設定して下さい。
出荷時のユニット番号は、0 番が設定されています。

2. インストール

この章では、本製品に関連する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

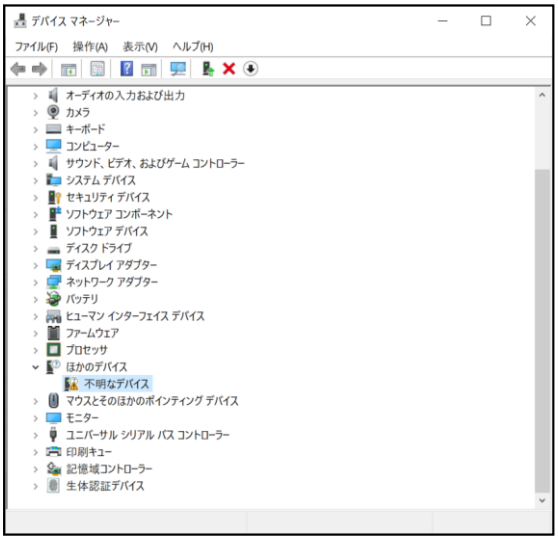
弊社ホームページより“MR540/MR580 用ソフトウェア”をダウンロードし、解凍して下さい。

2.2 USB ドライバのインストール

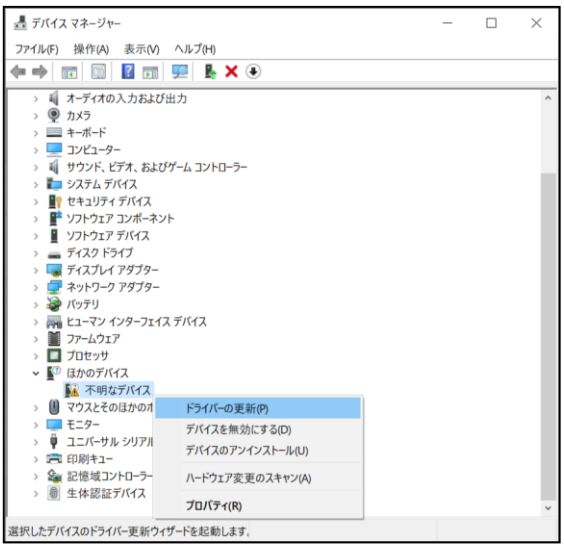
USB 通信により本製品を制御する場合は、USBドライバのインストールが必要です。USB 通信は、USB2.0 Full Speed で通信ができ、USB 電源供給方式は、セルフパワー方式です。本製品に DC24V 電源が、投入されると認識されます。

USBドライバは、“USB_Driver”フォルダに保存され、その後 32 ビット版 PC の場合は”32bit_Driver”フォルダを指定し、64 ビット版 PC の場合は”64bit_Driver”フォルダを指定して下さい。

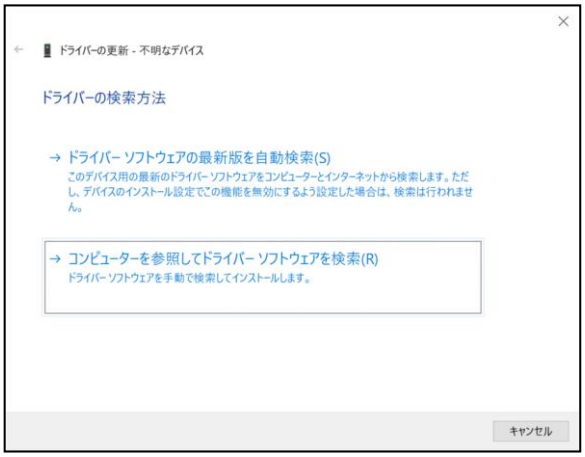
① デバイスマネージャー画面を開いて、[不明なデバイス]を選択します。



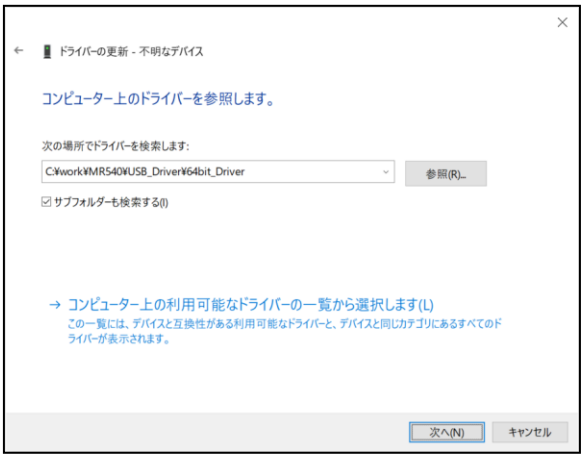
② 右クリックをし、[ドライバーの更新]を選択します。



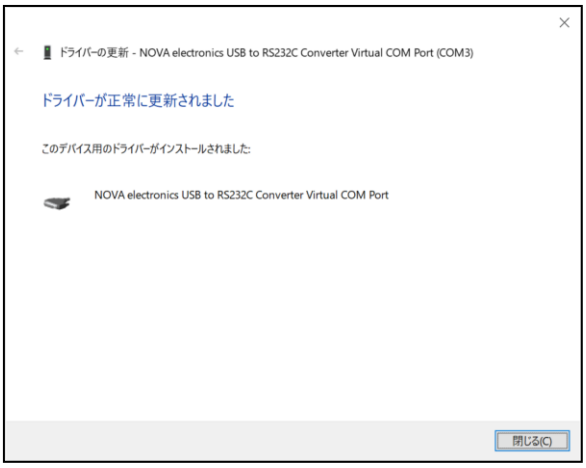
③ 「ドライバーの更新」画面で、[コンピュータを参照してドライバーソフトウェアを検索(R)]をクリックします。



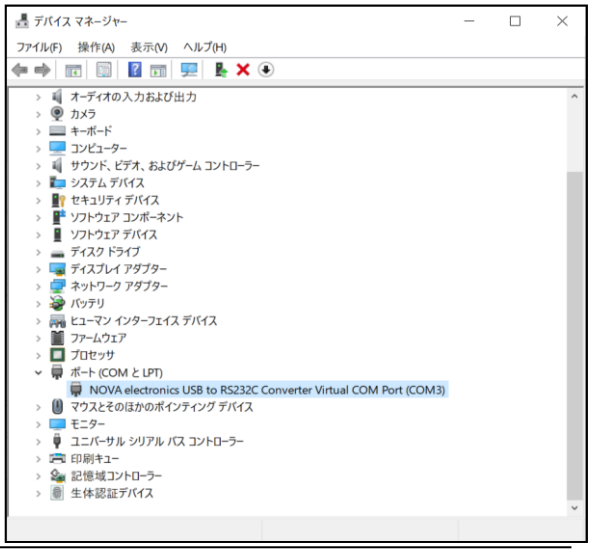
④ USB ドライバが保存されているフォルダを選択し、[次へ(N)]をクリックします。



⑤ インストールが正常に完了すると、右の画面が表示されます。[閉じる(C)] をクリックし、インストールを終了します。



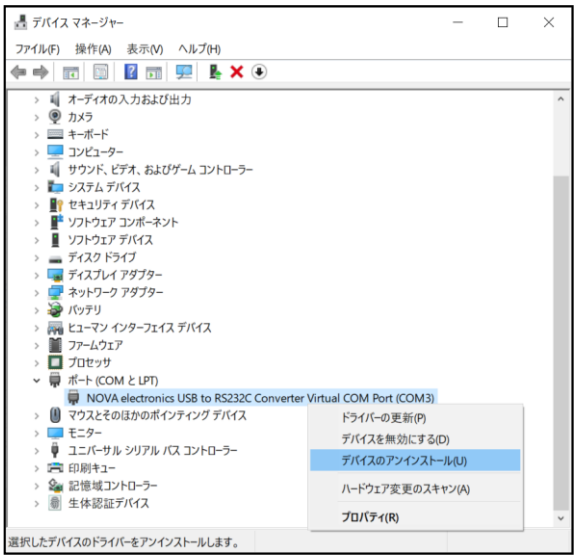
⑥ USB ドライバが、正常にインストールされた事を確認します。デバイスマネージャーの[ポート(COM と LPT)]に、[NOVA electronics USB to RS232C Converter Virtual COM Port (COMX)] (X:COM 番号)がある事を確認して下さい。表示されない場合は、再度インストールを行ってください。



2.3 USB ドライバの削除

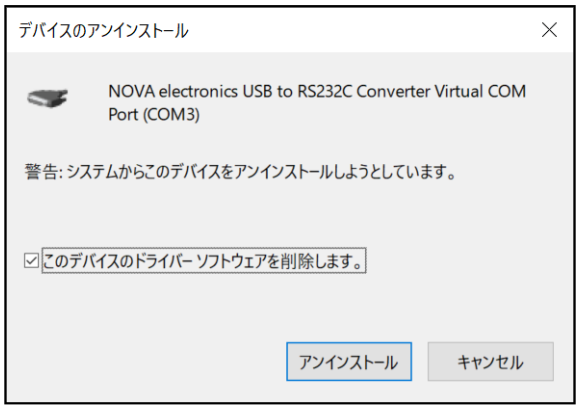
USBドライバを削除するには、対応の装置を接続しWindowsのデバイスマネージャから対応のCOM番号を選択し、削除します。複数台の削除の場合は、装置毎に削除操作が必要です。

① 削除するUSBドライバ対応の本製品をPCに接続します。デバイスマネージャ画面を開き、対応の COM 番号を選択します。

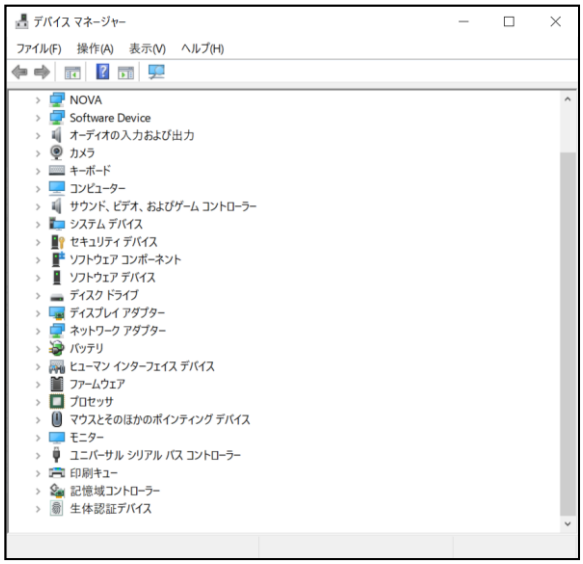


② 対応の COM 番号を選択したまま右クリックをし、[デバイスのアンインストール(U)]を選択します。

③ 「デバイスのアンインストール」画面で、[このデバイスのドライバーソフトウェアを削除します。]にチェックをいれ、[アンインストール]をクリックします。



④ デバイスマネージャーで指定の COM 番号が削除され、USBドライバも削除されていることを確認します。



2.4 LAN 通信の設定

LAN 通信（Ethernet）により本製品を制御する場合は、本製品に搭載の LAN PHY チップがネットワークプロトコル処理を行うため、複雑な設定や処理をすることなく、高速なネットワーク通信ができます。

a. DHCP サーバのある環境

DHCP サーバがある環境では、IP アドレスを自動取得いたしますので、設定は不要です。手動での設定はできませんのでご注意ください。MAC アドレスは、LAN コネクタに貼付されています。

b. DHCP サーバのない環境

DHCP サーバのない環境では、固定の IP アドレスが付与されますので、PC 側も固定の IP アドレスに設定します。
また、固定 IP アドレスのため、この場合は本製品 1 台のみの制御になります。接続方法は、本製品と PC をクロス LAN ケーブルで直接接続するか、またはスイッチングハブ（AUTO-MDIX 機能付）を使用して接続します。

【MR540/MR580】

DHCP サーバがない場合の固定 IP アドレス	
IP アドレス	192.168. 1. 11
サブネットマスク	255.255.255. 0
デフォルトゲートウェイ	192.168. 1. 1

LAN クロスケーブル

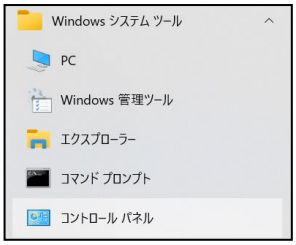
【PC】

固定 IP アドレス設定
例）192.168.1.10

［ DHCP サーバのない環境における PC･IP アドレス設定例 ］

① 設定

[Windows システムツール]－[コントロールパネル]をクリックします。

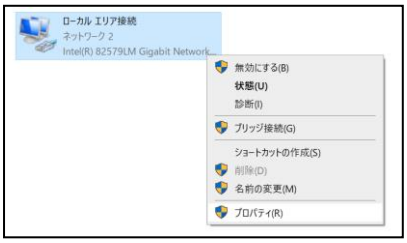


[ネットワークとインターネット]をクリックします。

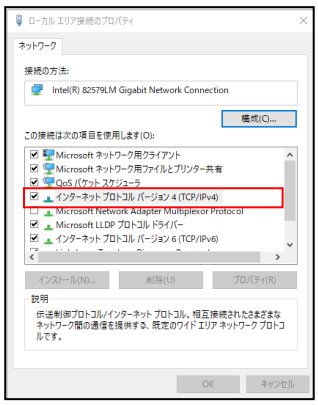
[ネットワークの状態とタスクの表示]をクリックします。

[アダプターの設定の変更]をクリックします。

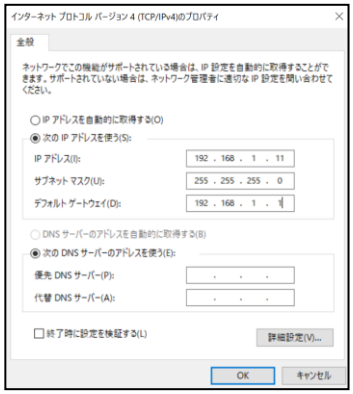
[ローカル　エリア接続]を右クリックし、[プロパティ(R)]をクリックします。



[インターネット　プロトコルバージョン4]を選択し、[OK]をクリックします。

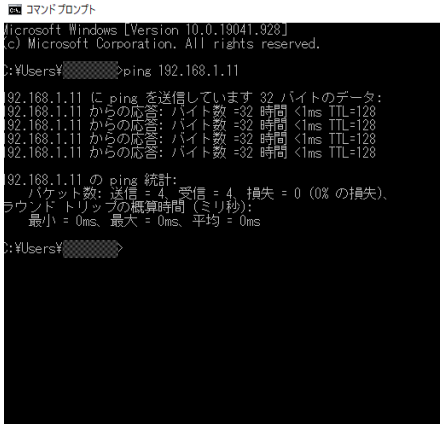


[次のアドレスを使う(S)]にチェック、IP アドレスを入力後、[OK]をクリックします。



②　　確認

コマンドプロンプト画面より「ping 192.168.1.11」を入力して応答がある事を確認します。



2.5 DLL の保存と登録

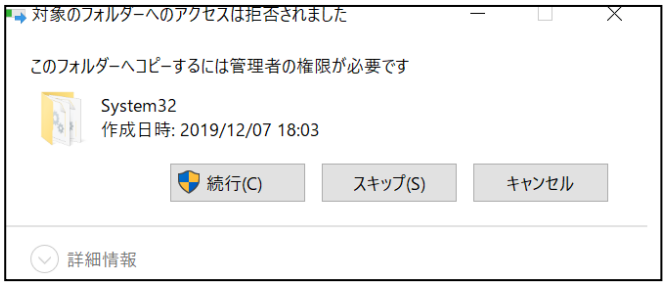
MR540_DLL.dll は、次のいずれかのフォルダに保存(コピー)して下さい。なお、Windows OS･32bit 版には 32bit 用 MR540_DLL.dll を、Windows OS･64bit 版には 64bit 用 MR540_DLL.dll をコピーして下さい。

a. 実行するアプリケーションの exe ファイルが保存されているフォルダ
b. Windows¥System32 フォルダ

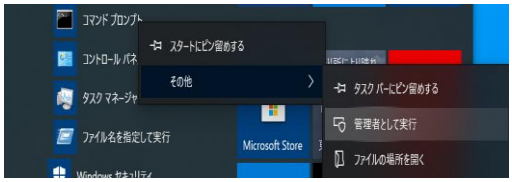
上記 b フォルダに保存する場合は、次の手順でシステムの登録設定を行ってください。

① MR540_DLL.dll を¥Windows¥System32 フォルダに保存します。

② 保存(コピー)する際に、右図のようにアクセス許可を尋ねるダイアログが表示されたら[続行(C)]をクリックして下さい。



③ 画面左下の[スタート]メニューから、[Windows システムツール]→[コマンドプロンプト]を選択します。右クリックし、[管理者として実行]を選択して下さい。

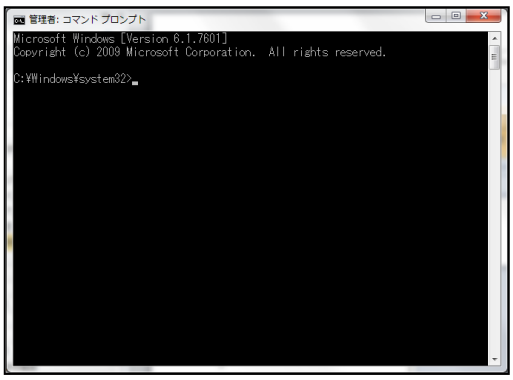


④ ユーザーアカウント制御画面が表示されたら、[はい]をクリックします。

⑤ コマンドプロンプト画面で、「system32>」の後に次のコマンドを入力して下さい。

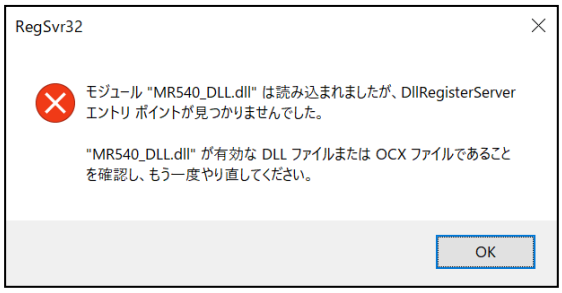
「regsvr32␣ MR540_DLL.dll」(␣:スペース)

入力後、Enter キーを押下して下さい



注意:「system32>」が、表示されない場合は、
ディレクトリをコマンドで変更して下さい。
(例: cd c:¥Windows¥system32 [Enter]押下)

⑥　右図の画面が表示されますが、[OK]をクリックして下さい。



2.6 DLL の削除

MR540_DLL.dll が、次のどちらに保存されているか確認して下さい。

a. 実行するアプリケーションの exe ファイルが保存されているフォルダ
b. Windows¥System32 フォルダ

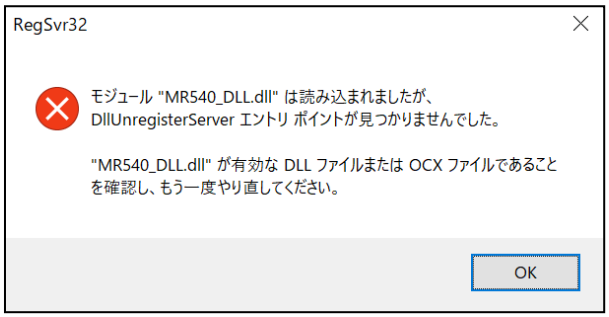
上記 a フォルダに保存されている場合は、MR540_DLL.dll ファイルをフォルダから削除します。
上記 b フォルダに保存されている場合は、次の手順でシステムの登録設定を解除してから、ファイルを削除して下さい。

① 登録設定の解除は、前項の登録と同じ手順でコマンドプロンプト画面を表示します。次に、「system32>」の後に、次のコマンドを入力してください。

「regsvr32␣/u␣MR540_DLL.dll」（␣:スペース）

入力後、Enter キーを押下して下さい。

② 右図のメッセージが出力されますが、[OK]をクリックして下さい。



③ MR540_DLL.dll ファイルを保存のフォルダから削除して下さい。

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。

3.1 開発環境

MR540/MR580 デバイスドライバは、Windows 10 (32ビット版、64 ビット版)に対応しています。
対応言語は、次の通りです。

対応言語
Microsoft Visual C++
Microsoft Visual Basic

3.2 ソフトウェア構成

項　目	フォルダ	ファイル(又はフォルダ)	説　明
デバイス ドライバ	Driver¥32Driver	MR540_DLL.dll	ダイナミックリンクライブラリ WindowsOS 32bit 用
	Driver¥64Driver	MR540_DLL.dll	ダイナミックリンクライブラリ WindowsOS 64bit 用
ライブラリ	LIB¥VC	MR540_DLL.h	MR540_DLL を使用するためのヘッダファイル(関数宣言、各定義) VC++専用
	LIB¥VC¥32LIB	MR540_DLL.lib	MR540_DLL を使用するためのライブラリファイル VC++専用 WindowsOS 32bit 用
	LIB¥VC¥64LIB	MR540_DLL.lib	MR540_DLL を使用するためのライブラリファイル VC++専用 WindowsOS 64bit 用
	LIB¥VB	MR540_DLL.vb	MR540_DLL を使用するための Declare 宣言、各定義ファイル VB 専用
評価ツール	Tool¥Tool¥32Tool	MR540_Tool.exe	MR540/MR580 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。 WindowsOS 32bit 用
	Tool¥IPTool ¥32IPTool	MR540_IPTool.exe	連続補間ドライブのデバッグツール 連続補間データを転送して保存、実行、セグメント毎の動作を確認できる。 WindowsOS 32bit 用
		Sample_data	補間データサンプルファイル: このファイルの補間データをを転送すると、データ読み出し、補間実行、データ削除ができる。
	Tool¥Tool ¥64Tool	MR540_Tool.exe	MR540/MR580 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。 WindowsOS 64bit 用
	Tool¥IPTool ¥64IPTool	MR540_IPTool.exe	連続補間ドライブのデバッグツール 連続補間データを転送して保存、実行、セグメント毎の動作を確認できる。 WindowsOS 64bit 用
		Sample_data	補間データサンプルファイル: このファイルの補間データを転送すると、データ読み出し、補間実行、データ削除ができる。

サンプル プログラム	Sample¥VC2015	Sample A	定量ドライブ、軸ステータス表示、論理位置カウンタ表示
		Sample B	連続補間ドライブの基本処理
		SampleB ¥Sample_data	補間データサンプルファイル：2CIP_2ST+CCW.ini SampleB 用の補間データ。VCSampleB.exe と同じフォルダに保存して使用する。
	Sample¥VB2015	Sample A	定量ドライブ、軸ステータス表示、論理位置カウンタ表示
		Sample B	連続補間ドライブの基本処理
		SampleB ¥Sample_data	補間データサンプルファイル：2CIP_2ST+CCW.ini SampleB 用の補間データ。VBSampleB.exe と同じフォルダに保存して使用する。

3.3 開発手順

3.3.1 VC++の場合

VC++でプログラミングを行う場合、以下のファイルが必要です。

WindowsOS が 32bit のとき

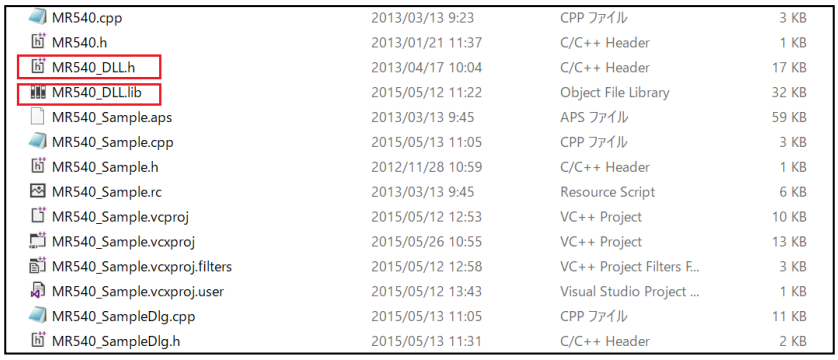
フォルダ	ファイル	説　明
LIB¥VC	MR540_DLL.h	MR540_DLL を使用するためのヘッダファイル(関数宣言、各定義)
LIB¥VC¥32LIB	MR540_DLL.lib	MR540_DLL を使用するためのライブラリ
Driver¥32Driver	MR540_DLL.dll	ダイナミックリンクライブラリ

WindowsOS が 64bit のとき

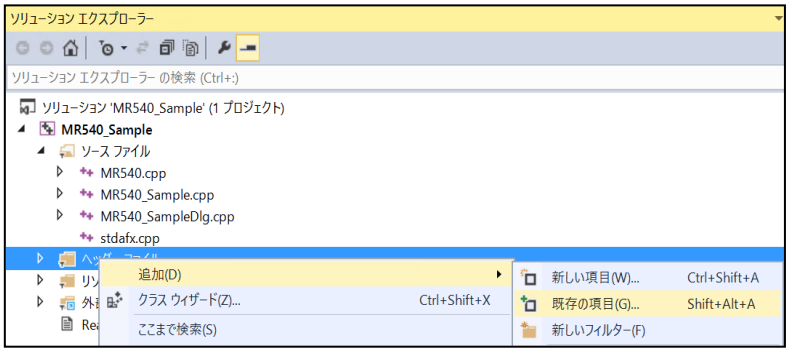
フォルダ	ファイル	説　明
LIB¥VC	MR540_DLL.h	MR540_DLL を使用するためのヘッダファイル(関数宣言、各定義)
LIB¥VC¥64LIB	MR540_DLL.lib	MR540_DLL を使用するためのライブラリ
Driver¥64Driver	MR540_DLL.dll	ダイナミックリンクライブラリ

以下の手順でファイルをプロジェクトに設定して下さい。(以下は Visual Studio 2015 での手順を示しています。)

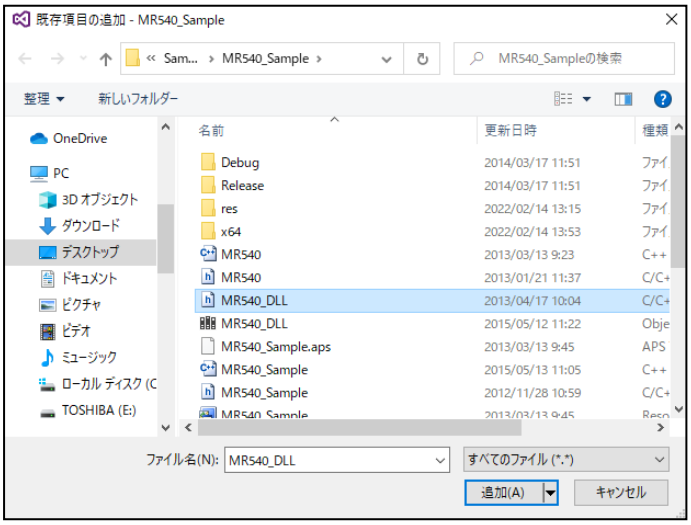
① LIB¥VC の MR540_DLL.lib と MR540_DLL.h の2つのファイルを、開発するプロジェクトのフォルダにコピーします。
(MR540_ProjectDlg.cpp 等の C 言語ソースファイルが保存されているフォルダ)にコピーします。



② VC++総合開発環境にて、ヘッダファイル MR540_DLL.h をプロジェクトに追加します。
[表示]－[ソリューションエクスプローラー]を選択し、ソリューションエクスプローラーが表示されたら、プロジェクト名の上で右クリックをして、[追加]－[既存の項目]を選択します。

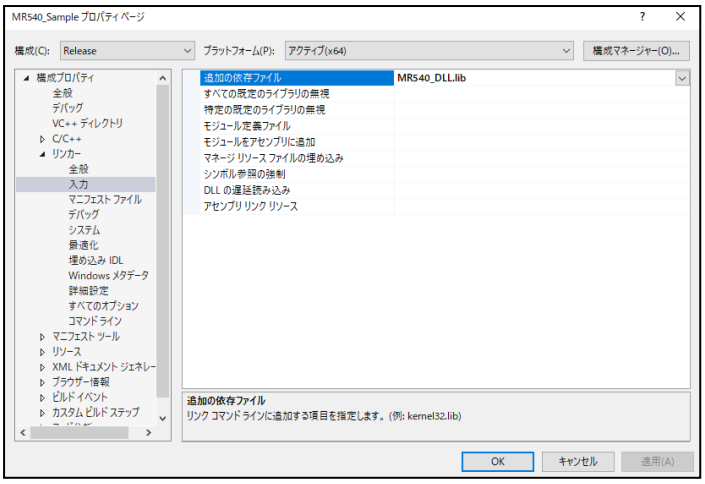


MR540_DLL.h を選択し、[追加(A)]をクリックします。



③ API 関数を使用するソースファイルに MR540_DLL.h をインクルードします。

④ ライブラリーMR540_DLL.lib を追加します。
[プロジェクト]－[プロパティ]を選択し、プロパティページを表示します。
[構成プロパティ]－[リンカ]－[入力]を選択し、[追加の依存ファイル]に MR540_DLL.lib を追加します。



⑤ ダイナミックリンクライブラリ MR540_DLL.dll をコピーします。

ダイナミックリンクライブラリを以下のいずれかの場所にコピーします。
但し、2. インストール 2.5DLL の保存と登録 で既に下記の表の b の場所にコピーを行っている場合は、この作業は必要ありません。

a	プロジェクトがあるフォルダ	①項で MR540_DLL.lib、MR540_DLL.h をコピーしたフォルダと同じフォルダに 32bit 用または 64bit 用 MR540_DLL.dllをコピー
b	Windows¥System32 フォルダ	32bit 用または 64bit 用 MR540_DLL.dll をコピー

注意事項:
・Windows¥System32 に登録する場合、コピーだけでなく、DLL をシステムに登録させる必要があります。詳しくは、2.5 DLL の保存と登録 または Microsoft 社のホームページをご覧下さい。
・DLL を複数の場所にコピーした場合には、DLL の更新または削除時に作業漏れのないように、DLL の保存場所、バージョン等の管理には十分ご注意下さい。

⑥ 「3.4 API」の関数を使用しプログラミングを行って下さい。

3.3.2 VB の場合

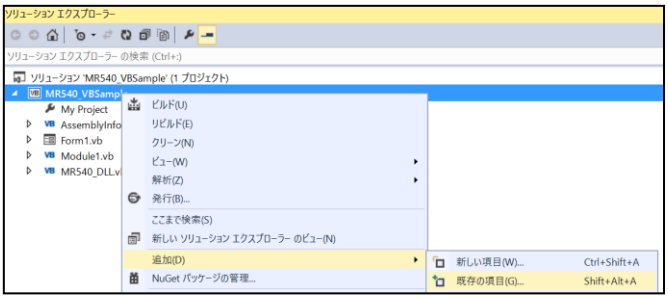
VB でプログラミングを行う場合、以下のファイルが必要です。

フォルダ	ファイル	説明
Lib¥VB	MR540_DLL.vb	MR540_DLL を使用するための Declare 宣言、各定義ファイル
Driver¥32Driver	MR540_DLL.dll	ダイナミックリンクライブラリ（WindowsOS が 32bit のとき）
Driver¥64Driver	MR540_DLL.dll	ダイナミックリンクライブラリ（WindowsOS が 64bit のとき）

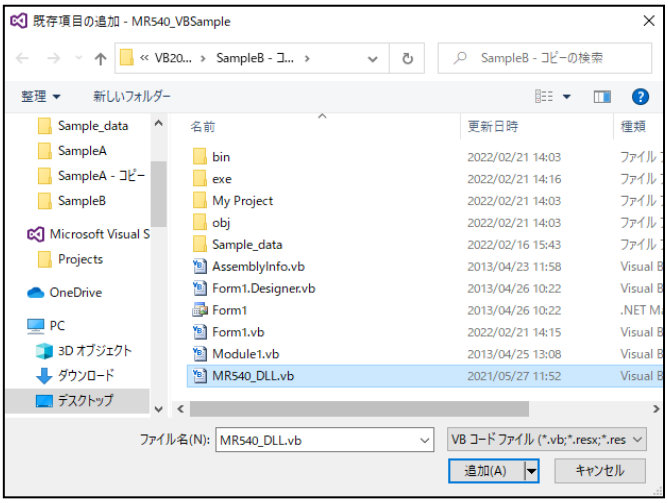
以下の手順でファイルをプロジェクトに設定して下さい。(以下は Visual Studio 2015 での手順を示しています。)

⑦ Lib¥VB フォルダに入っている MR540_DLL.vb ファイルを開発するアプリケーションのフォルダにコピーします。

⑧ vb ファイル MR540_DLL.vb をプロジェクトに追加します。
[表示]－[ソリューションエクスプローラ]を選択し、ソリューションエクスプローラが表示されたら、プロジェクト名の上で右クリックをし、[追加]－[既存項目の追加]を選択します。



⑨ MR540_DLL.vb を選択し、[追加(A)]をクリックします。



⑩ ダイナミックリンクライブラリ MR540_DLL.dll をコピーします。

ダイナミックリンクライブラリを以下のいずれかの場所にコピーします。
但し、2. インストール　2.5 DLL の保存と登録　で既に下記の b の場所にコピーを行っている場合は、この作業は必要ありません。

a	プロジェクトがあるフォルダの bin フォルダ	①項で MR540_DLL.lib、MR540_DLL.h をコピーしたフォルダと同じフォルダに 32bit 用または 64bit 用 MR540_DLL.dllをコピー
b	C:¥Windows¥System32 フォルダ	32bit 用または 64bit 用 MR540_DLL.dll をコピー

⑪ 「3.4 API」の関数を使用しプログラミングを行って下さい。

注意：VBでは、本ユニットに関する割り込みを使用することはできません。

3.4 ＡＰＩ

MR540_DLL.dll が、アプリケーションに提供する API について説明します。
レジスタの内容詳細・機能詳細につきましては、MCX314As/AL 取扱説明書を参照して下さい。

【 関数一覧 】

◆主な引数の指定方法について

(1) 通信インターフェイス定義

定義は、次の通りです。

VC			
#define	NCI_IFTYPE_USB	0x00	USB
#define	NCI_IFTYPE_LAN	0x01	LAN
VB			
Public Const	NCI_IFTYPE_USB	&H0s	USB
Public Const	NCI_IFTYPE_LAN	&H1s	LAN

(2) ユニット番号

ロータリスイッチで設定されます。(16 進数：0～9,A,B,C,D,E,F)

ロータリスイッチと同じ番号をユニット番号として使用します。

ユニット番号を API 関数から指定する場合は、10 進数で指定します。(10 進数：0～9,10,11,12,13,14,15)

(3) IC 番号

メイン基板の MCX314AL を'0'、増設 4 軸基板の MCX314AL を'1'とします。

MR540 は、メイン基板のみですが、IC 番号は必ず'0'を指定して下さい。

(4) 軸定義

軸定義は、次の通りです。

VC			
#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_ALL	0xF	全軸
VB			
Public Const	AXIS_X	&H1s	X軸
Public Const	AXIS_Y	&H2s	Y軸
Public Const	AXIS_Z	&H4s	Z軸
Public Const	AXIS_U	&H8s	U軸
Public Const	AXIS_ALL	&HFs	全軸

① 1 軸指定の場合： AXIS_X, AXIS_Y, AXIS_Z, AXIS_U のいずれかを指定します。
(使用例) X軸にドライブ速度 1000 を設定する。 Nmc_Speed(No, IcNo, AXIS_X, 1000);

② 複数軸指定の場合： ビットＯＲ演算子を使用します。
(使用例) X,Y,Z 軸に設定する。 Nmc_Speed(No, IcNo, AXIS_X | AXIS_Y | AXIS_Z, 1000);

③ 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。 Nmc_Speed(No, IcNo, AXIS_ALL, 1000);

(5) モータコントロール IC(MCX314AL)のレジスタ定義

MCX314AL のライトレジスタとリードレジスタからの定義は次の通りです。

VC							
#define	MCX_WR0	0x0000	コマンドレジスタ	#define	MCX_RR0	0x0000	主ステータスレジスタ
#define	MCX_WR1	0x0001	モードレジスタ1 (軸毎)	#define	MCX_RR1	0x0001	ステータスレジスタ1 (軸毎)
#define	MCX_WR2	0x0002	モードレジスタ2 (軸毎)	#define	MCX_RR2	0x0002	ステータスレジスタ2 (軸毎)
#define	MCX_WR3	0x0003	モードレジスタ3 (軸毎)	#define	MCX_RR3	0x0003	ステータスレジスタ3 (軸毎)
#define	MCX_WR4	0x0004	アウトプットレジスタ	#define	MCX_RR4	0x0004	インプットレジスタ1
#define	MCX_WR5	0x0005	補間モードレジスタ	#define	MCX_RR5	0x0005	インプットレジスタ2
#define	MCX_WR6	0x0006	ライトデータレジスタ1	#define	MCX_RR6	0x0006	リードレジスタ1
#define	MCX_WR7	0x0007	ライトデータレジスタ2	#define	MCX_RR7	0x0007	リードレジスタ2

VB			
Public Const MCX_WR0 &H0s	コマンドレジスタ	Public Const MCX_RR0 &H0s	主ステータスレジスタ
Public Const MCX_WR1 &H1s	モードレジスタ1 （軸毎）	Public Const MCX_RR1 &H1s	ステータスレジスタ1 （軸毎）
Public Const MCX_WR2 &H2s	モードレジスタ 2 （軸毎）	Public Const MCX_RR2 &H2s	ステータスレジスタ2 （軸毎）
Public Const MCX_WR3 &H3s	モードレジスタ 3 （軸毎）	Public Const MCX_RR3 &H3s	ステータスレジスタ3 （軸毎）
Public Const MCX_WR4 &H4s	アウトプットレジスタ	Public Const MCX_RR4 &H4s	インプットレジスタ1
Public Const MCX_WR5 &H5s	補間モードレジスタ	Public Const MCX_RR5 &H5s	インプットレジスタ2
Public Const MCX_WR6 &H6s	ライトデータレジスタ1	Public Const MCX_RR6 &H6s	リードレジスタ1
Public Const MCX_WR7 &H7s	ライトデータレジスタ2	Public Const MCX_RR7 &H7s	リードレジスタ2

(6) モータコントロール IC(MCX314AL)の命令定義

MCX314AL の主な命令は次の通りです。

VC		
ドライブ命令		
#define CMD_F_DRV_P	0x20	+方向定量ﾊﾟﾙｽﾄﾞﾗｲﾌﾞ
#define CMD_F_DRV_M	0x21	-方向定量ﾊﾟﾙｽﾄﾞﾗｲﾌﾞ
#define CMD_C_DRV_P	0x22	+方向連続ﾊﾟﾙｽﾄﾞﾗｲﾌﾞ
#define CMD_C_DRV_M	0x23	-方向連続ﾊﾟﾙｽﾄﾞﾗｲﾌﾞ
#define CMD_START_HOLD	0x24	ﾄﾞﾗｲﾌﾞ開始ﾎｰﾙﾄﾞ
#define CMD_START_FREE	0x25	ﾄﾞﾗｲﾌﾞ開始ﾌﾘｰ
#define CMD_STP_STS_CLR	0x25	終了ｽﾃｰﾀｽｸﾘｱ
#define CMD_STOP_DEC	0x26	ﾄﾞﾗｲﾌﾞ減速停止
#define CMD_STOP_SUDDEN	0x27	ﾄﾞﾗｲﾌﾞ即停止
その他の命令		
#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出力
#define CMD_SYNC_ACTIVE	0x65	同期動作起動
#define CMD_NOP	0x0F	NOP （軸切り換え用）
アプリケーションから制御する補間ドライブ命令		
#define CMD_IP_2ST	0x30	2 軸直線補間ﾄﾞﾗｲﾌﾞ
#define CMD_IP_3ST	0x31	3 軸直線補間ﾄﾞﾗｲﾌﾞ
#define CMD_IP_CW	0x32	CW 円弧補間ﾄﾞﾗｲﾌﾞ
#define CMD_IP_CCW	0x33	CCW 円弧補間ﾄﾞﾗｲﾌﾞ
#define CMD_IP_2BP	0x34	2 軸 BP 補間ﾄﾞﾗｲﾌﾞ
#define CMD_IP_3BP	0x35	3 軸 BP 補間ﾄﾞﾗｲﾌﾞ
#define CMD_BP_ENABLED	0x36	BP ﾚｼﾞｽﾀ書き込み可
#define CMD_BP_DISABLED	0x37	BP ﾚｼﾞｽﾀ書き込み不可
#define CMD_BP_STACK	0x38	BP ﾃﾞｰﾀｽﾀｯｸ
#define CMD_BP_CLR	0x39	BP ﾃﾞｰﾀｸﾘｱ
#define CMD_IP_1STEP	0x3A	補間ｼﾝｸﾞﾙｽﾃｯﾌﾟ
#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

VB		
ドライブ命令		
Public Const CMD_F_DRV_P	&H20s	+方向定量ﾊﾟﾙｽﾄﾞﾗｲﾌﾞ
Public Const CMD_F_DRV_M	&H21s	-方向定量ﾊﾟﾙｽﾄﾞﾗｲﾌﾞ
Public Const CMD_C_DRV_P	&H22s	+方向連続ﾊﾟﾙｽﾄﾞﾗｲﾌﾞ
Public Const CMD_C_DRV_M	&H23s	-方向連続ﾊﾟﾙｽﾄﾞﾗｲﾌﾞ
Public Const CMD_START_HOLD	&H24s	ﾄﾞﾗｲﾌﾞ開始ﾎｰﾙﾄﾞ
Public Const CMD_START_FREE	&H25s	ﾄﾞﾗｲﾌﾞ開始ﾌﾘｰ
Public Const CMD_STP_STS_CLR	&H25s	終了ｽﾃｰﾀｽｸﾘｱ
Public Const CMD_STOP_DEC	&H26s	ﾄﾞﾗｲﾌﾞ減速停止
Public Const CMD_STOP_SUDDEN	&H27s	ﾄﾞﾗｲﾌﾞ即停止
その他の命令		
Public Const CMD_HOME_EXEC	&H62s	自動原点出し実行
Public Const CMD_DEVCTR_CLR	&H63s	偏差ｶｳﾝﾀｸﾘｱ出力
Public Const CMD_SYNC_ACTIVE	&H65s	同期動作起動
Public Const CMD_NOP	&H0Fs	NOP（軸切り換え用）
アプリケーションから制御する補間ドライブ命令		
Public Const CMD_IP_2ST	&H30s	2 軸直線補間ﾄﾞﾗｲﾌﾞ
Public Const CMD_IP_3ST	&H31s	3 軸直線補間ﾄﾞﾗｲﾌﾞ
Public Const CMD_IP_CW	&H32s	CW　円弧補間ﾄﾞﾗｲﾌﾞ
Public Const CMD_IP_CCW	&H33s	CCW 円弧補間ﾄﾞﾗｲﾌﾞ
Public Const CMD_IP_2BP	&H34s	2 軸 BP 補間ﾄﾞﾗｲﾌﾞ
Public Const CMD_IP_3BP	&H35s	3 軸 BP 補間ﾄﾞﾗｲﾌﾞ
Public Const CMD_BP_ENABLED	&H36s	BP ﾚｼﾞｽﾀ書き込み可
Public ConstCMD_BP_DISABLED	&H37s	BP ﾚｼﾞｽﾀ書き込み不可
Public Const CMD_BP_STACK	&H38s	BP ﾃﾞｰﾀｽﾀｯｸ
Public Const CMD_BP_CLR	&H39s	BP ﾃﾞｰﾀｸﾘｱ
Public Const CMD_IP_1STEP	&H3As	補間ｼﾝｸﾞﾙｽﾃｯﾌﾟ
Public Const CMD_IP_DEC_VALID	&H3Bs	減速有効
Public Const CMD_IP_DEC_INVALID	&H3Cs	減速無効
Public Const CMD_IP_INTRPT_CLR	&H3Ds	補間割り込みｸﾘｱ

3.4.1 基本関数

3.4.1.1	関数名	Nmc_Initialize	機能	DLLを初期化する

アプリケーションの最初に実行します。通信インターフェイスの指定と DLL の初期化処理を行います。

【構文】

VC	int Nmc_Initialize(int InterfaceType);
VB	Function Nmc_Initialize(ByVal InterfaceType As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	InterfaceType	int	通信インターフェイス種別 （USB または LAN）	入力
VB	InterfaceType	Integer	通信インターフェイス種別 （USB または LAN）	入力

【戻り値】

言語	型	内 容
VC	int	初期化した結果 NCI_DONE：正常 NCI_INV_ARG：パラメータ不正 NCI_ALREADY_INIT：初期化済み NCI_INIT_ERR：初期化失敗
VB	Integer	初期化した結果

【使用例】

VC	Status = Nmc_Initialize(NCI_IFTYPE_USB); // USB通信で、DLLを初期化処理
VB	Status = Nmc_Initialize(NCI_IFTYPE_USB) ' USB通信で、DLLを初期化処理

3.4.1.2	関数名	Nmc_Finalize	機能	DLLを終了する

アプリケーションの終了時に DLL の終了処理を行います。アプリケーションの最後に必ず実行して下さい。

【構文】

VC	int Nmc_ Finalize (void);
VB	Function Nmc_ Finalize ( ) As Integer

【入力パラメータ】 なし

【戻り値】

言語	型	内 容
VC	int	終了した結果 NCI_DONE：正常 NCI_NOT_INIT：未初期化
VB	Integer	終了した結果

【使用例】

VC	Status = Nmc_ Finalize ( ); // DLLの終了処理
VB	Status = Nmc_ Finalize ( ) ' DLLの終了処理

3．4．1．3	関数名	Nmc_Open	機能	ユニットの使用を開始する

指定のユニットに対して、割り込み処理の有無を指定してオープンします。

【構文】

VC	BOOL　　Nmc_Open(int No, BOOL IntrptFlg);
VB	Function　Nmc_Open(ByVal No As Integer, ByVal IntrptFlg As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IntrptFlg	BOOL	割り込みの使用を指定する。 TRUE： 使用する。FALSE： 使用しない。	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IntrptFlg	Integer	割り込みの未使用を指定する。 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)	入力

【戻り値】

言語	型	内　容
VC	BOOL	TRUE :オープンに成功、 FALSE :オープンに失敗
VB	Integer	オープンに成功すると0以外、失敗すると0

【使用例】

VC	Status = Nmc_Open(0, FALSE);　　// ユニット番号0をオープン、割り込みは使用しない
VB	Status = Nmc_Open(0, False)　　' ユニット番号0をオープン、割り込みは使用しない

3．4．1．4	関数名	Nmc_Close	機能	ユニットの使用を終了する

指定のユニットをクローズします。

【構文】

VC	BOOL　　Nmc_Close(int No);
VB	Function　Nmc_Close(ByVal No As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
VB	No	Integer	ユニット番号(0～15)	入力

【戻り値】

言語	型	内　容
VC	BOOL	TRUE:クローズに成功、 FALSE:クローズに失敗
VB	Integer	クローズにに成功すると0以外、失敗すると0

【使用例】

VC	Status = Nmc_Close(0);　　// ユニット番号0をクローズ
VB	Status = Nmc_Close(0)　　' ユニット番号0をクローズ

3．4．1．5	関数名	Nmc_CloseAll	機能	全てのユニットの使用を終了する

オープン中の全てのユニットをクローズします。

【構文】

VC	BOOL　Nmc_CloseAll(void);
VB	Function Nmc_CloseAll() As Integer

【入力パラメータ】 なし

【戻り値】

言語	型	内 容
VC	BOOL	TRUE:クローズに成功、 FALSE:クローズに失敗
VB	Integer	クローズに成功すると0以外、失敗すると0

【使用例】

VC	Status = Nmc_CloseAll(); // 全てのユニットをクローズ
VB	Status = Nmc_CloseAll() ' 全てのユニットをクローズ

3．4．1．6	関数名	Nmc_WriteReg	機能	ライトレジスタ(WR0～WR7)にデータを書き込む

本関数には軸指定がないため、軸指定の必要な WR1,2,3 レジスタ書き込みにおいては、直前に書き込んだ命令の軸に対して、書き込まれます。

【構文】

VC	void　Nmc_WriteReg(int No, int IcNo, int RegNum, long Dat);
VB	Sub　Nmc_WriteReg(ByVal No As Integer, ByVal IcNo As Integer, ByVal RegNum As Integer,　ByVal Dat As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	RegNum	int	書き込むレジスタ(MCX_WR0～MCX_WR7)	入力
	Dat	long	書き込むデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	RegNum	Integer	書き込むレジスタ(MCX_WR0～MCX_WR7)	入力
	Dat	Integer	書き込むデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteReg(1, 0, MCX_WR0, 0x8000);　　//ユニット1のIC0のWR0に0x8000書き込み=ソフトリセット
VB	Call Nmc_WriteReg(1, 0, MCX_WR0, &H8000)　' ユニット1のIC0のWR0に0x8000書き込み=ソフトリセット

3．4．1．7	関数名	Nmc_ReadReg	機能	リードレジスタ(RR0～RR7)からデータを読み出す

本関数には、軸指定がないため、軸指定の必要な RR1,2 レジスタ読み出しにおいては、直前に書き込んだ命令の軸に対して、読み出しされます。

【構文】

VC	long　　Nmc_ReadReg(int No, int IcNo, int RegNum);
VB	Function Nmc_ReadReg(ByVal No As Integer, ByVal IcNo As Integer, ByVal RegNum As Integer)As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	RegNum	int	読み出すレジスタ(MCX_RR0～MCX_RR7)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	RegNum	Integer	読み出すレジスタ(MCX_RR0～MCX_RR7)	入力

【戻り値】

言語	型	内　容
VC	long	リードレジスタから読み出したデータ
VB	Integer	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);　　// IC0のリードレジスタRR0の読み出し
VB	Data = Nmc_ReadReg(No, 0, MCX_RR0)　　'　IC0のリードレジスタRR0の読み出し

3．4．1．8	関数名	Nmc_SetEvent	機能	割り込みを処理するユーザー関数を設定する

本関数の設定を行うと、割り込みが発生した時に設定の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。設定解除は、Nmc_ResetEvent で行います。

【構文】

VC	BOOL Nmc_SetEvent(int No, int IcNo,LPTHREAD_START_ROUTINE UserThread, LPVOID lpParameter);
VB	使用できません

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	UserThread	LPTHREAD_START_ROUTINE	ユーザー関数のアドレス	入力
	lpParameter	LPVOID	ユーザー関数スレッドに渡す1つの引数を指定する。	入力

【戻り値】

言語	型	内　容
VC	BOOL	TRUE:設定に成功、 FALSE:設定に失敗

【使用例】

VC	Stetus = Nmc_SetEvent(0, 1, MC_EventFunc0, lpParam); //　ユニット0, IC1に関数のアドレスと引数を設定 Nmc_WriteReg1(0, 0, AXIS_ALL, 0x8000);　　// IC0のドライブ停止時に、割り込み発生の設定（全軸）

3．4．1．9	関数名	Nmc_ResetEvent	機能	割り込みを処理するユーザー関数の設定を解除する

本関数の設定を行うと、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。

【構文】

VC	BOOL Nmc_ResetEvent(int No, int IcNo);
VB	使用できません

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力

【戻り値】

言語	型	内 容
VC	BOOL	TRUE:解除に成功、 FALSE:解除に失敗

【使用例】

VC	Status = Nmc_ResetEvent(0, 0); // ユニット0、IC0に設定したユーザー関数を解除する

3．4．1．10	関数名	Nmc_ReadEvent	機能	各軸の割り込み発生要因(RR3レジスタ値)を取得する

割り込み発生要因(MCX314AL の RR3 レジスタ値)の確認の際に、この関数を使用します。
この関数では、それまでに発生した割り込み要因が OR 条件で取得できます。この関数を呼び出した後は、その内容がクリアされます。従って、以前に発生した割り込み要因は、確認できませんのでご注意ください。
ユーザー関数の設定(Nmc_SetEvent/ Nmc_ResetEvent)に関わらず、割り込み発生要因は、本関数が実行されるまで、保存されています。同じ割り込みが複数回発生した場合は、上書きとなります。

【構文】

VC	BOOL Nmc_ReadEvent(int No, int IcNo, int* Rr3X, int* Rr3Y, int* Rr3Z, int* Rr3U);
VB	使用できません

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Rr3X	int	X軸のRR3を格納する為のバッファへのポインタ	出力
	Rr3Y	int	Y軸のRR3を格納する為のバッファへのポインタ	出力
	Rr3Z	int	Z軸のRR3を格納する為のバッファへのポインタ	出力
	Rr3U	int	U軸のRR3を格納する為のバッファへのポインタ	出力

【戻り値】

言語	型	内 容
VC	BOOL	TRUE:取得に成功、 FALSE:取得に失敗

【使用例】

VC	int Rr3X[2], Rr3Y[2], Rr3Z[2], Rr3U[2];
	Nmc_ReadEvent(No, 0, &Rr3X[0], &Rr3Y[0], &Rr3Z[0], &Rr3U[0]); // IC0･RR3レジスタの読み出し
	Nmc_ReadEvent(No, 1, &Rr3X[1], &Rr3Y[1], &Rr3Z[1], &Rr3U[1]); // IC1･RR3レジスタの読み出し

3.4.2 リセット、命令関数

3.4.2.1	関数名	Nmc_Reset	機能	ユニットに搭載しているICをリセットする

指定の IC（MCX314AL)に対してソフトリセットを行います。
ソフトリセットを行いますと、パラメータ類も初期化されますので、パラメータ類の設定を改めて行ってください。

【構文】

VC	void Nmc_Reset(int No, int IcNo);
VB	Sub Nmc_Reset(ByVal No As Integer, ByVal IcNo As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力

【戻り値】 なし

【使用例】

VC	Nmc_Reset(0, 0); // ユニット番号0のIC0をリセット
VB	Call Nmc_Reset(0, 0) ' ユニット番号0のIC0をリセット

3.4.2.2	関数名	Nmc_Command	機能	指定軸の命令を実行する

指定 IC の WR0 レジスタに、指定軸への命令番号を書き込みます。

【構文】

VC	void Nmc_Command(int No, int IcNo, int Axis, int cmd);
VB	Sub Nmc_Command(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal cmd As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	命令を実行する軸（複数軸の指定可能）	入力
	cmd	int	命令番号 （3．4節（5）の「ドライブ命令」と「その他の命令」を設定）	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)。基本基板が0,オプション基板が1。	入力
	Axis	Integer	命令を実行する軸（複数軸の指定可能）	入力
	cmd	Integer	命令番号 （3．4節（5）の「ドライブ命令」と「その他の命令」を設定）	入力

【戻り値】 なし

【使用例】

VC	Nmc_Command(0, 0, AXIS_X, CMD_F_DRV_P); // ユニット番号0/IC0/X軸の＋方向定量ドライブ
VB	Call Nmc_Command(0, 0, AXIS_X, CMD_F_DRV_P) ' ユニット番号0/IC0/X軸の＋方向定量ドライブ

3.4.3 ライトレジスタ関数

3.4.3.1	関数名	Nmc_WriteReg0	機能	WR0(コマンドレジスタ)にデータを書き込む

指定 IC の WR0 レジスタにデータを書き込みます。

【構文】

VC	void Nmc_WriteReg0(int No, int IcNo, long wdata);
VB	Sub Nmc_WriteReg0(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	wdata	long	書き込むデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	wdata	Integer	書き込むデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteReg0(0, 0, 0x120); // ユニット番号0/IC0のX軸の＋方向定量ドライブの実行
VB	Call Nmc_WriteReg(0, 0, &H120) ' ユニット番号0/IC0のX軸の＋方向定量ドライブの実行

3.4.3.2	関数名	Nmc_WriteReg1	機能	WR1(モードレジスタ1)にデータを書き込む

指定軸の WR1 レジスタにデータを書き込みます。

【構文】

VC	void Nmc_WriteReg1(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_WriteReg1(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを書き込む軸	入力
	wdata	long	書き込むデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを書き込む軸	入力
	wdata	Integer	書き込むデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteReg1(0, 1, AXIS_X, 0x8000); // ユニット番号0/IC1のX軸のドライブエンド割り込み有効
VB	Call Nmc_WriteReg1(0, 1, AXIS_X, &H8000) ' ユニット番号0/IC1のX軸のドライブエンド割り込み有効

3．4．3．3	関数名	Nmc_WriteReg2	機能	WR2(モードレジスタ2)にデータを書き込む

指定軸の WR2 レジスタにデータを書き込みます。

【構文】

VC	void Nmc_WriteReg2(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_WriteReg2(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを書き込む軸	入力
	wdata	long	書き込むデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを書き込む軸	入力
	wdata	Integer	書き込むデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteReg2(1, 0, AXIS_Y, 0x2000); // ユニット番号1/IC0のY軸のALARM入力信号有効
VB	Call Nmc_WriteReg2(1, 0, AXIS_Y, &H2000) ' ユニット番号1/IC0のY軸のALARM入力信号有効

3．4．3．4	関数名	Nmc_WriteReg3	機能	WR3(モードレジスタ3)にデータを書き込む

指定軸の WR3 レジスタにデータを書き込みます。

【構文】

VC	void Nmc_WriteReg3(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_WriteReg3(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを書き込む軸	入力
	wdata	long	書き込むデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを書き込む軸	入力
	wdata	Integer	書き込むデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteReg3(1, 1, AXIS_Y, 0x0001); // ユニット番号1/IC1のY軸のマニュアル減速設定
VB	Call Nmc_WriteReg3(1, 1, AXIS_Y, &H1) ' ユニット番号1/IC1のY軸のマニュアル減速設定

3．4．3．5	関数名	Nmc_WriteReg4	機能	WR4(アウトプットレジスタ)にデータを書き込む

指定 IC の WR4 レジスタにデータを書き込みます。

【構文】

VC	void Nmc_WriteReg4(int No, int IcNo, long wdata);
VB	Sub Nmc_WriteReg4(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	wdata	long	書き込むデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	wdata	Integer	書き込むデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteReg4(2, 0, 0x0100); // ユニット番号2/IC0のZ軸汎用出力OUT0を出力ON
VB	Call Nmc_WriteReg4(2, 0, &H0100) ' ユニット番号2/IC0のZ軸汎用出力OUT0を出力ON

3．4．3．6	関数名	Nmc_WriteReg5	機能	WR5(補間モードレジスタ)にデータを書き込む

指定 IC の WR5 レジスタにデータを書き込みます。補間ドライブのモード設定で使用します。

【構文】

VC	void Nmc_WriteReg5(int No, int IcNo, long wdata);
VB	Sub Nmc_WriteReg5(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	wdata	long	書き込むデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	wdata	Integer	書き込むデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteReg5(2, 1, 0x0024); // ユニット番号2/IC1に補間軸設定（主軸：X、第2軸：Y、第3軸：Z）
VB	Call Nmc_WriteReg5(2, 1, &H0024) ' ユニット番号2/IC1に補間軸設定（主軸：X、第2軸：Y、第3軸：Z）

3．4．3．7	関数名	Nmc_WriteReg6	機能	WR6(ライトデータレジスタ1)にデータを書き込む

指定 IC の WR6 レジスタにデータを書き込みます。

【構文】

VC	void Nmc_WriteReg6(int No, int IcNo, long wdata);
VB	Sub Nmc_WriteReg6(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	wdata	long	書き込むデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	wdata	Integer	書き込むデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteReg6(3, 0, 0x1234); // ユニット番号3/IC0のライトデータ1にデータ1234hを書き込む
VB	Call Nmc_WriteReg6(3, 0, &H1234) ' ユニット番号3/IC0のライトデータ1にデータ1234hを書き込む

3．4．3．8	関数名	Nmc_WriteReg7	機能	WR7(ライトデータレジスタ2)にデータを書き込む

指定 IC の WR7 レジスタにデータを書き込みます。

【構文】

VC	void Nmc_WriteReg7(int No, int IcNo, long wdata);
VB	Sub Nmc_WriteReg7(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	wdata	long	書き込むデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	wdata	Integer	書き込むデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteReg7(3, 0, 0x5678); // ユニット番号3/IC0のライトデータ2にデータ5678hを書き込む
VB	Call Nmc_WriteReg7(3, 0, &H5678) ' ユニット番号3/IC0のライトデータ2にデータ5678hを書き込む

3.4.4 リードレジスタ関数

3.4.4.1	関数名	Nmc_ReadReg0	機能	RRO(主ステータスレジスタ)のデータを読み出す

指定 IC の RR0 レジスタのデータを読み出します。

【構文】

VC	long Nmc_ReadReg0(int No, int IcNo);
VB	Function Nmc_ReadReg0(ByVal No As Integer, ByVal IcNo As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力

【戻り値】

言語	型	内 容
VC	long	RRO(主ステータスレジスタ)のデータ
VB	Integer	RRO(主ステータスレジスタ)のデータ

【使用例】

VC	Data = Nmc_ReadReg0(1, 0); // ユニット番号1/IC0の主ステータスレジスタのデータを読み出す
VB	Data = Nmc_ReadReg0(1, 0) ' ユニット番号1/IC0の主ステータスレジスタのデータを読み出す

3.4.4.2	関数名	Nmc_ReadReg1	機能	RR1(ステータスレジスタ1)のデータを読み出す

指定軸の RR1 レジスタのデータを読み出します。

【構文】

VC	long Nmc_ReadReg1(int No, int IcNo, int Axis);
VB	Function Nmc_ReadReg1(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを読み出す軸	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを読み出す軸	入力

【戻り値】

言語	型	内 容
VC	long	RR1(ステータスレジスタ1)のデータ
VB	Integer	RR1(ステータスレジスタ1)のデータ

【使用例】

VC	Data = Nmc_ReadReg1(1, 0, AXIS_X); // ユニット番号1/IC0/X軸のステータスレジスタ1のデータを読み出す
VB	Data = Nmc_ReadReg1(1, 0, AXIS_X) ' ユニット番号1/IC0/X軸のステータスレジスタ1のデータを読み出す

3．4．4．3	関数名	Nmc_ReadReg2	機能	RR2(ステータスレジスタ2)のデータを読み出す

指定軸の RR2 レジスタのデータを読み出します。

【構文】

VC	long　　Nmc_ReadReg2(int No, int IcNo, int Axis);
VB	Function Nmc_ReadReg2(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを読み出す軸	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを読み出す軸	入力

【戻り値】

言語	型	内　容
VC	long	RR2(ステータスレジスタ2)のデータ
VB	Integer	RR2(ステータスレジスタ2)のデータ

【使用例】

VC	Data = Nmc_ReadReg2(2, 1, AXIS_Y);　// ユニット番号2/IC1/Y軸のステータスレジスタ2のデータを読み出す
VB	Data = Nmc_ReadReg2(2, 1, AXIS_Y)　'　ユニット番号2/IC1/Y軸のステータスレジスタ2のデータを読み出す

3．4．4．4	関数名	Nmc_ReadReg4	機能	RR4(インプットレジスタ1)のデータを読み出す

指定 IC の RR4 レジスタのデータを読み出します。

【構文】

VC	long　　Nmc_ReadReg4(int No, int IcNo);
VB	Function Nmc_ReadReg4(ByVal No As Integer, ByVal IcNo As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力

【戻り値】

言語	型	内　容
VC	long	RR4(インプットレジスタ1)のデータ （入力信号状態とレジスタ表示(0/1)の関係は、本取扱説明書4.3節を参照）
VB	Integer	RR4(インプットレジスタ1)のデータ

【使用例】

VC	Data = Nmc_ReadReg4(2, 1);　　// ユニット番号2/IC1インプットレジスタ1のデータを読み出す
VB	Data = Nmc_ReadReg4(2, 1)　　'　ユニット番号2/IC1インプットレジスタ1のデータを読み出す

3．4．4．5	関数名	Nmc_ReadReg5	機能	RR5(インプットレジスタ2)のデータを読み出す

指定 IC の RR5 レジスタのデータを読み出します。

【構文】

VC	long　　Nmc_ReadReg5(int No, int IcNo);
VB	Function Nmc_ReadReg5(ByVal No As Integer, ByVal IcNo As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力

【戻り値】

言語	型	内　容
VC	long	RR5(インプットレジスタ2)のデータ （入力信号状態とレジスタ表示(0/1)の関係は、本取扱説明書4.3節を参照）
VB	Integer	RR5(インプットレジスタ2)のデータ

【使用例】

VC	Data = Nmc_ReadReg5(3, 0);　　// ユニット番号3/IC0インプットレジスタ2のデータを読み出す
VB	Data = Nmc_ReadReg5(3, 0)　　' ユニット番号3/IC0インプットレジスタ2のデータを読み出す

3．4．4．6	関数名	Nmc_ReadReg6	機能	RR6(リードデータレジスタ1)のデータを読み出す

指定 IC の RR6 レジスタのデータを読み出します。

【構文】

VC	long　　Nmc_ReadReg6(int No, int IcNo);
VB	Function Nmc_ReadReg6(ByVal No As Integer, ByVal IcNo As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力

【戻り値】

言語	型	内　容
VC	long	RR6(リードデータレジスタ1)のデータ
VB	Integer	RR6(リードデータレジスタ1)のデータ

【使用例】

VC	Data = Nmc_ReadReg6(3, 1);　　// ユニット番号3/IC1のリードレジスタ1のデータを読み出す
VB	Data = Nmc_ReadReg6(3, 1)　　' ユニット番号3/IC1のリードレジスタ1のデータを読み出す

3．4．4．7	関数名	Nmc_ReadReg7	機能	RR7(リードデータレジスタ2)のデータを読み出す

指定 IC の RR7 レジスタのデータを読み出します。

【構文】

VC	long　　Nmc_ReadReg7(int No, int IcNo);
VB	Function Nmc_ReadReg7(ByVal No As Integer, ByVal IcNo As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力

【戻り値】

言語	型	内　容
VC	long	RR7(リードデータレジスタ2)のデータ
VB	Integer	RR7(リードデータレジスタ2)のデータ

【使用例】

VC	Data = Nmc_ReadReg7(3, 1);　　// ユニット番号3/IC1のリードレジスタ2のデータを読み出す
VB	Data = Nmc_ReadReg7(3, 1)　　' ユニット番号3/IC1のリードレジスタ2のデータを読み出す

3.4.5　パラメータ設定関数

3.4.5.1	関数名	Nmc_Range	機能	レンジを設定する

指定軸の速度倍率を決定するためのレンジを設定します。

【構文】

VC	void Nmc_Range(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_Range(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_Range(0, 1, AXIS_ALL, 800000);　　// ユニット番号0/IC1/全軸のレンジに800000(速度倍率10)を設定
VB	Call Nmc_Range(0, 1, AXIS_ALL, 800000)　' ユニット番号0/IC1/全軸のレンジに800000(速度倍率10)を設定

3.4.5.2	関数名	Nmc_Jerk	機能	加速度増加率(加加速度)を設定する

指定軸において S 字加減速を使用する場合の加速度増加率を設定します。

【構文】

VC	void Nmc_Jerk(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_Jerk(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal wdata As Long)
VB	Sub Nmc_Jerk(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Long	ユニット番号(0～15)	入力
	IcNo	Long	IC番号(0,1)	入力
	Axis	Long	データを設定する軸	入力
	wdata	Long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_Jerk(0, 1, AXIS_ALL, 1000);　　// ユニット番号0/IC1/全軸に加速度増加率1000を設定
VB	Call Nmc_Jerk(0, 1, AXIS_ALL, 1000)　' ユニット番号0/IC1/全軸に加速度増加率1000を設定

3．4．5．3	関数名	Nmc_Acc	機能	加速度を設定する

指定軸の加速度を設定します。

【構文】

VC	Void Nmc_Acc(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_Acc(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_Acc(2, 0, AXIS_X, 100); // ユニット番号2/IC0/X軸に加速度100を設定
VB	Call Nmc_Acc(2, 0, AXIS_X, 100) ' ユニット番号2/IC0/X軸に加速度100を設定

3．4．5．4	関数名	Nmc_Dec	機能	減速度を設定する

指定軸の減速度を設定します。

【構文】

VC	void Nmc_Dec(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_Dec(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_Dec(2, 1, AXIS_X, 200); // ユニット番号2/IC1/X軸に減速度200を設定
VB	Call Nmc_Dec(2, 1, AXIS_X, 200) ' ユニット番号2/IC1/X軸に減速度200を設定

3．4．5．5	関数名	Nmc_StartSpd	機能	初速度を設定する

指定軸の初速度を設定します。

【構文】

VC	void Nmc_StartSpd(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_StartSpd(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_StartSpd(3, 0, AXIS_Y, 100); // ユニット番号3/IC0/Y軸に初速度100を設定
VB	Call Nmc_StartSpd(3, 0, AXIS_Y, 100) ' ユニット番号3/IC0/Y軸に初速度100を設定

3．4．5．6	関数名	Nmc_Speed	機能	ドライブ速度を設定する

指定軸のドライブ速度を設定します。

【構文】

VC	void Nmc_Speed(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_Speed(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_Speed(3, 1, AXIS_Z, 1000); // ユニット番号3/IC1/Z軸にドライブ速度1000を設定
VB	Call Nmc_Speed(3, 1, AXIS_Z, 1000) ' ユニット番号3/IC1/Z軸にドライブ速度1000を設定

3．4．5．7	関数名	Nmc_Pulse	機能	出力パルス数を設定する （VC専用）

指定軸の定量パルスドライブの出力パルス数を設定します。(符号無 32 ビットデータ)
アプリケーションから制御する単独の直線補間ドライブ、円弧補間ドライブにおいては、各軸の終点を設定します。終点座標は、現在座標に対する相対値で指定します。(符号有 32 ビットデータ)

【構文】

VC	void Nmc_Pulse(int No, int IcNo, int Axis, long wdata);
VB	使用できません

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_Pulse(4, 0, AXIS_U, 10000); // ユニット番号4/IC0/U軸に出力パルス数10000を設定

3．4．5．8	関数名	Nmc_Pulse_VB	機能	出力パルス数を設定する （VB専用）

指定軸の定量パルスドライブの出力パルス数を設定します。（符号無 32 ビットデータ）
アプリいケーションから制御する単独の直線補間ドライブ、円弧補間ドライブにおいては、各軸の終点を設定します。終点座標は、現在座標に対する相対値で指定します。（符号有 32 ビットデータ）

【構文】

VC	使用できません
VB	Sub Nmc_Pulse_VB(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Double)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Double	設定するデータ	入力

【戻り値】 なし

【使用例】

VB	Call Nmc_Pulse(4, 0, AXIS_U, 10000)　　' ユニット番号4/IC0/U軸に出力パルス数10000を設定

3．4．5．9	関数名	Nmc_DecP	機能	マニュアル減速点を設定する(VC専用)

指定軸においてマニュアル減速モードを使用する場合の減速点を設定します。

【構文】

VC	void Nmc_DecP(int No, int IcNo, int Axis, ULONG wdata);
VB	使用できません

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	ULONG	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_DecP(4, 1, AXIS_X, 30000);　　// ユニット番号4/IC1/X軸にマニュアル減速点30000を設定

3．4．5．10	関数名	Nmc_DecP_VB	機能	マニュアル減速点を設定する （VB専用）

指定軸においてマニュアル減速モードを使用する場合の減速点を設定します。

【構文】

VC	使用できません
VB	Sub Nmc_DecP_VB(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Double)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Double	設定するデータ	入力

【戻り値】 なし

【使用例】

VB	Call Nmc_DecP(4, 1, AXIS_X, 30000)　　// ユニット番号4/IC1/X軸にマニュアル減速点30000を設定

3．4．5．11	関数名	Nmc_Center	機能	円弧中心点を設定する

円弧補間ドライブの際の中心点を、指定軸に設定します。

【構文】

VC	void Nmc_Center(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_Center(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC 番号(0,1)	入力
	Axis	int	データを設定する軸。	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC 番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_Center(5, 0, AXIS_Y, 1500);　　// ユニット番号 5/IC0/Y 軸に円弧中心点 1500 を設定
VB	Call Nmc_Center(5, 0, AXIS_Y, 1500)　'　ユニット番号 5/IC0/Y 軸に円弧中心点 1500 を設定

3．4．5．12	関数名	Nmc_Lp	機能	論理位置カウンタを設定する

指定軸の論理位置カウンタ値を設定します。

【構文】

VC	void Nmc_Lp(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_Lp(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_Lp(5, 1, AXIS_ALL, 0); // ユニット番号5/IC1/全軸の論理位置カウンタを0クリア
VB	Call Nmc_Lp(5, 1, AXIS_ALL, 0) ' ユニット番号5/IC1/全軸の論理位置カウンタを0クリア

3．4．5．13	関数名	Nmc_Ep	機能	実位置カウンタを設定する

指定軸の実位置カウンタ値を設定します。

【構文】

VC	void Nmc_Ep(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_Ep(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_Ep(6, 0, AXIS_ALL, 0); // ユニット番号6/IC0/全軸の実位置カウンタを0クリア
VB	Call Nmc_Ep(6, 0, AXIS_ALL, 0) ' ユニット番号6/IC0/全軸の実位置カウンタを0クリア

3．4．5．14	関数名	Nmc_CompP	機能	COMP＋レジスタを設定する

指定軸の COMP+レジスタ(論理/実位置カウンタと大小比較をするレジスタ)に値を設定します。

【構文】

VC	void Nmc_CompP(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_CompP(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_CompP(6, 1, AXIS_Z, 50000); // ユニット番号6/IC1/Z軸のCOMP+レジスタに50000を設定
VB	Call Nmc_CompP(6, 1, AXIS_Z, 50000) ' ユニット番号6/IC1/Z軸のCOMP+レジスタに50000を設定

3．4．5．15	関数名	Nmc_CompM	機能	COMP－レジスタを設定する

指定軸の COMP－レジスタ(論理/実位置カウンタと大小比較をするレジスタ)に値を設定します。

【構文】

VC	void Nmc_CompM(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_CompM(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_CompM(7, 0, AXIS_Z, -50000); // ユニット番号7/IC0/Z軸のCOMP-レジスタに-50000を設定
VB	Call Nmc_CompM(7, 0, AXIS_Z, -50000) ' ユニット番号7/IC0/Z軸のCOMP-レジスタに-50000を設定

3．4．5．16	関数名	Nmc_AccOfst	機能	加速カウンタオフセットを設定する

加速カウンタオフセット値は、定量パルスドライブの減速時の出力パルス数の補正を行います。
補正の必要な指定軸の加速カウンタオフセット値を設定します。
リセット時は、8 がセットされています。

【構文】

VC	void Nmc_AccOfst(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_AccOfst(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_AccOfst(7, 1, AXIS_U, 20); // ユニット番号7/IC1/U軸の加速カウンタオフセットに20を設定
VB	Call Nmc_AccOfst(7, 1, AXIS_U, 20) ' ユニット番号7/IC1/U軸の加速カウンタオフセットに20を設定

3．4．5．17	関数名	Nmc_DJerk	機能	減速度増加率を設定する

指定軸において S 字加減速を使用する場合の減速度増加率を設定します。

【構文】

VC	void Nmc_DJerk(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_DJerk(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_DJerk(8, 0, AXIS_U, 1000); // ユニット番号8/IC0/U軸の減速度増加率に1000を設定
VB	Call Nmc_DJerk(8, 0, AXIS_U, 1000) ' ユニット番号8/IC0/U軸の減速度増加率に1000を設定

3．4．5．18	関数名	Nmc_HomeSpd	機能	原点検出速度を設定する

自動原点出しのステップ2、ステップ3の低速サーチ速度を設定します。

【構文】

VC	void Nmc_HomeSpd(int No, int IcNo, int Axis, long wdata);
VB	Sub Nmc_HomeSpd(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_HomeSpd(8, 1, AXIS_U, 200); // ユニット番号8/IC1/U軸の原点検出速度に200を設定
VB	Call Nmc_HomeSpd(8, 1, AXIS_U, 200) ' ユニット番号8/IC1/U軸の原点検出速度に200を設定

3.4.6　各種モード設定関数

3.4.6.1	関数名	Nmc_ExpMode	機能	拡張モードを設定する

指定軸の拡張モード設定では、自動原点出しモードの設定やフィルタ機能の設定、その他の機能の詳細な設定を行います。
自動原点出しモード設定の詳細は、本書の 4.4 節、MCX314As/AL 取扱説明書の 2.5 節を参照して下さい。
フィルタ機能設定の詳細は、本書の 4.8 節、MCX314As/AL 取扱説明書の 2.8 節、6.16 節を参照して下さい。
その他の機能設定の詳細は、MCX314As/AL 取扱説明書の 6.16 節を参照して下さい。

【構文】

VC	void Nmc_ExpMode(int No, int IcNo, int Axis, long EM6_data, long EM7_data);
VB	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)

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	EM6_data	long	拡張モードレジスタ EM6 に設定するデータ	入力
	EM7_data	long	拡張モードレジスタ EM7 に設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	EM6_data	Integer	拡張モードレジスタ EM6 に設定するデータ	入力
	EM7_data	Integer	拡張モードレジスタ EM7 に設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_ExpMode(0, 1, AXIS_X, 0x7700, 0x0045);　　// ユニット番号0/IC1/X軸の拡張モードを設定 EM6_data:全信号フィルタ有効・時定数3、 EM7_data:自動原点出しステップ1,2,4実行
VB	Call Nmc_ExpMode(0, 1, AXIS_X, &H7700, &H0045)　' ユニット番号0/IC1/X軸の拡張モードを設定 EM6_data:全信号フィルタ有効・時定数3、 EM7_data:自動原点出しステップ1,2,4実行

3．4．6．2	関数名	Nmc_SyncMode	機能	同期動作モードを設定する

指定軸の同期動作モード設定では、MCX314AL に同期動作機能の設定を行います。
詳細は、本書の 4.7 節、MCX314As/AL 取扱説明書の 2.6 節、6.18 節を参照して下さい。

【構文】

VC	void Nmc_SyncMode(int No, int IcNo, int Axis, long SM6_data, long SM7_data);
VB	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)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを設定する軸	入力
	SM6_data	long	同期動作モードレジスタ SM6 に設定するデータ	入力
	SM7_data	long	同期動作モードレジスタ SM7 に設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	SM6_data	Integer	同期動作モードレジスタ SM6 に設定するデータ	入力
	SM7_data	Integer	同期動作モードレジスタ SM7 に設定するデータ	入力

【戻り値】 なし

【使用例】

VC	// ユニット番号1/IC0において同期動作（X軸ドライブ終了時に、Y軸＋方向定量ドライブ開始）の設定を行う。
	// ① X軸同期動作モードの設定：他軸起動=Y軸、起動要因=E-END、動作（自軸）=なし
	// ② Y軸同期動作モードの設定：他軸起動=無、 起動要因=無、 、動作=＋方向定量ドライブ
	① Nmc_SyncMode(1, 0, AXIS_X, 0x2020, 0);
	② Nmc_SyncMode(1, 0, AXIS_Y, 0, 0x0001);

VB	' ユニット番号1/IC0において同期動作（X軸ドライブ終了時に、Y軸＋方向定量ドライブ開始）の設定を行う。 ' ① X軸同期動作モードの設定：他軸起動=Y軸、起動要因=E-END、動作（自軸）=なし ' ② Y軸同期動作モードの設定：他軸起動=無、 起動要因=無、 、動作=＋方向定量ドライブ ① Call Nmc_SyncMode(1, 0, AXIS_X, &H2020, 0) ② Nmc_SyncMode(1, 0, AXIS_Y, 0, &H0001)

3.4.7　データ読み出し関数

3.4.7.1	関数名	Nmc_ReadLp	機能	論理位置カウンタを読み出す

指定軸の論理位置カウンタの現在値を読み出します。

【構文】

VC	long　　Nmc_ReadLp(int No, int IcNo, int Axis);
VB	Function Nmc_ReadLp(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを読み出す軸	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを読み出す軸	入力

【戻り値】

言語	型	内 容
VC	long	現在の論理位置カウンタの値
VB	Integer	現在の論理位置カウンタの値

【使用例】

VC	Data = Nmc_ReadLp(1, 0, AXIS_X);　　// ユニット番号1/IC0/X軸の論理位置カウンタを読み出す
VB	Data = Nmc_ReadLp(1, 0, AXIS_X)　　' ユニット番号1/IC0/X軸の論理位置カウンタを読み出す

3.4.7.2	関数名	Nmc_ReadEp	機能	実位置カウンタを読み出す

指定軸の実位置カウンタの現在値を読み出します。

【構文】

VC	long　　Nmc_ReadEp(int No, int IcNo, int Axis);
VB	Function Nmc_ReadEp(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを読み出す軸	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを読み出す軸	入力

【戻り値】

言語	型	内 容
VC	long	現在の実位置カウンタの値
VB	Integer	現在の実位置カウンタの値

【使用例】

VC	Data = Nmc_ReadEp(1, 0, AXIS_Y);　　// ユニット番号1/IC0/Y軸の実位置カウンタを読み出す
VB	Data = Nmc_ReadEp(1, 0, AXIS_Y)　　' ユニット番号1/IC0/Y軸の実位置カウンタを読み出す

3．4．7．3	関数名	Nmc_ReadSpeed	機能	現在ドライブ速度を読み出す

指定軸のドライブ速度の現在値を読み出します。

【構文】

VC	long　　Nmc_ ReadSpeed (int No, int IcNo, int Axis);
VB	Function Nmc_ ReadSpeed(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを読み出す軸	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを読み出す軸	入力

【戻り値】

言語	型	内　容
VC	long	現在ドライブ速度
VB	Integer	現在ドライブ速度

【使用例】

VC	Data = Nmc_ ReadSpeed (1, 0, AXIS_Z);　　// ユニット番号1/IC0/Z軸の現在ドライブ速度を読み出す
VB	Data = Nmc_ ReadSpeed (1, 0, AXIS_Z)　　'　ユニット番号1/IC0/Z軸の現在ドライブ速度を読み出す

3．4．7．4	関数名	Nmc_ReadAccDec	機能	現在加／減速度を読み出す

指定軸の現在加速度値、または現在減速度値を読み出します。
ドライブ停止時の読み出しデータは、不定値です。

【構文】

VC	long　　Nmc_ReadAccDec(int No, int IcNo, int Axis);
VB	Function Nmc_ReadAccDec(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを読み出す軸	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを読み出す軸	入力

【戻り値】

言語	型	内　容
VC	long	現在加／減速度
VB	Integer	現在加／減速度

【使用例】

VC	Data = Nmc_ReadAccDec(1, 0, AXIS_U);　　// ユニット番号1/IC0/U軸の現在加/減速度を読み出す
VB	Data = Nmc_ReadAccDec(1, 0, AXIS_U)　　'　ユニット番号1/IC0/U軸の現在加/減速度を読み出す

3．4．7．5	関数名	Nmc_ReadSyncBuff	機能	同期バッファレジスタを読み出す

指定軸の同期バッファレジスタ(同期動作機能で使用するレジスタ)値を読み出します。

【構文】

VC	long　　Nmc_ReadSyncBuff(int No, int IcNo, int Axis);
VB	Function Nmc_ReadSyncBuff(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを読み出す軸	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを読み出す軸	入力

【戻り値】

言語	型	内　容
VC	long	同期バッファレジスタの値
VB	Integer	同期バッファレジスタの値

【使用例】

VC	Data = Nmc_ReadSyncBuff(1, 1, AXIS_X);　　// ユニット番号1/IC1/X軸の同期バッファレジスタを読み出す
VB	Data = Nmc_ReadSyncBuff(1, 1, AXIS_X)　　' ユニット番号1/IC1/X軸の同期バッファレジスタを読み出す

3.4.8 状態取得関数

3.4.8.1	関数名	Nmc_GetDriveStatus	機能	ドライブ状態を取得する

指定軸のドライブ状態(ドライブ中／終了)を取得します。

【構文】

VC	int Nmc_GetDriveStatus(int No, int IcNo, int Axis);
VB	Function Nmc_GetDriveStatus(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	ドライブ状態を取得する軸(複数軸の指定可能)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	ドライブ状態を取得する軸(複数軸の指定可能)	入力

【戻り値】

言語	型	内 容
VC	int	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。
VB	Integer	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(2, 0, AXIS_Y) = = 0)
	　　AfxMessageBox("ユニット番号2/IC0/Y軸ドライブ終了");
	else
	　　AfxMessageBox("ユニット番号2/IC0/Y軸ドライブ中");

VB	If Nmc_GetDriveStatus(2, 0, AXIS_Y) = 0 Then
	　　Call MsgBox("ユニット番号2/IC0/Y軸ドライブ終了")
	Else
	　　Call MsgBos("ユニット番号2/IC0/Y軸ドライブ中")
	End If

3．4．8．2	関数名	Nmc_OptionCheck	機能	増設4軸基板実装の確認

増設 4 軸基板の実装を確認します。

【構文】

VC	int　　Nmc_OptionCheck (int No);
VB	Function Nmc_OptionCheck (ByVal No As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
VB	No	Integer	ユニット番号(0～15)	入力

【戻り値】

言語	型	内 容
VC	int	確認の結果 NMC_EXIST_IC：増設 4 軸基板あり NMC_NO_IC：増設 4 軸基板なし
VB	Long	確認の結果
VB	Integer	確認の結果

【使用例】

VC	Status = Nmc_OptionCheck (4);　　// ユニット番号4の増設4軸基板の有無チェック
VB	Status = Nmc_OptionCheck (4)　　' ユニット番号4の増設4軸基板の有無チェック

3.4.9　書き込み・読み出し関数

3．4．9．1	関数名	Nmc_WriteRegSetAxis	機能	指定ライトレジスタ(WR1～WR3)にデータを書き込む

指定軸の指定したライトレジスタ(WR1～WR3)にデータを書き込みます。

【構文】

VC	void Nmc_WriteRegSetAxis(int No, int IcNo, int Axis, int RegNum, long wdata);
VB	Sub Nmc_WriteRegSetAxis(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal RegNum As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを書き込む軸(複数軸指定可能)	入力
	RegNum	int	データを書き込むライトレジスタの番号	入力
	wdata	long	書き込むデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを書き込む軸(複数軸指定可能)	入力
	RegNum	Integer	データを書き込むライトレジスタの番号	入力
	wData	Inreger	書き込むデータ	入力

【戻り値】　なし

【使用例】

VC	Nmc_WriteRegSetAxis(0, 0, AXIS_ALL, MCX_WR2, 0x2000); //ユニット0, IC0の全軸のWR2レジスタにALARM有効(2000)Hを書き込む
VB	Call Nmc_WriteRegSetAxis(0, 0, AXIS_ALL, MCX_WR2, &H2000) '　ユニット0, IC0の全軸のWR2レジスタにALARM有効(2000)Hを書き込む

3．4．9．2	関数名	Nmc_ReadRegSetAxis	機能	指定リードレジスタ(RR1,RR2)のデータを読み出す

指定軸の指定したリードレジスタ(RR1, RR2)のデータを読み出します。

【構文】

VC	long　　Nmc_ReadRegSetAxis(int No, int IcNo, int Axis, int RegNum);
VB	Function Nmc_ReadRegSetAxis(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal RegNum As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを読み出す軸	入力
	RegNum	int	データを読み出すリードレジスタの番号	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを読み出す軸	入力
	RegNum	Integer	データを読み出すリードレジスタの番号	入力

【戻り値】

言語	型	内 容
VC	long	指定軸の指定したリードレジスタのデータ
VB	Integer	指定軸の指定したリードレジスタのデータ

【使用例】

VC	Data = Nmc_ReadRegSetAxis(0, 1, AXIS_X, MCX_RR1); // ユニット0, IC1のX軸のRR1レジスタのデータを読み出す
VB	Data = Nmc_ReadRegSetAxis(0, 1, AXIS_X, MCX_RR1) '　ユニット0, IC1のX軸のRR1レジスタのデータを読み出す

3．4．9．3	関数名	Nmc_WriteData	機能	命令コードによりパラメータ･データを書き込む

データ書き込み命令コードにより、指定軸にパラメータ･データを書き込みます。

【構文】

VC	void Nmc_WriteData(int No, int IcNo, int Axis, int cmd, long wdata);
VB	Sub Nmc_WriteData(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal cmd As Integer, ByVal wdata As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを書き込む軸（複数軸の指定可能）	入力
	cmd	int	データ書き込み命令コード((00)H～(0E)H, (61)H) （次頁の【注3．4．9】を参照）	入力
	wdata	long	設定するデータ	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを書き込む軸（複数軸の指定可能）	入力
	cmd	Integer	データ書き込み命令コード((00)H～(0E)H, (61)H) （次頁の【注3．4．9】を参照）	入力
	wdata	Integer	設定するデータ	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteData(1, 0, AXIS_ALL, 0x05, 1000); // ユニット1、IC0の全軸にドライブ速度（命令ｺｰﾄﾞ(05)H）1000を設定する
VB	Call Nmc_WriteData(1, 0, AXIS_ALL, &H05, 1000) ' ユニット1、IC0の全軸にドライブ速度（命令ｺｰﾄﾞ(05)H）1000を設定する

3．4．9．4	関数名	Nmc_WriteData2	機能	命令コードにより拡張モードまたは同期動作モードを設定する

データ書き込み命令コードにより、指定軸に拡張モード、または同期動作モードを設定します。

【構文】

VC	void Nmc_WriteData2(int No, int IcNo, int Axis, int cmd, long WR6_data, long WR7_data);
VB	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)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを書き込む軸（複数軸の指定可能）	入力
	cmd	int	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	入力
	WR6_data	long	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	入力
	WR7_data	long	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	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを書き込む軸(複数軸の指定可能）	入力
	cmd	Integer	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	入力
	WR6_data	Integer	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	入力
	WR7_data	Integer	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	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteData2(1, 1, AXIS_X, 0x60, 0x5F00, 0x0045); // ユニット1, IC1のX軸の拡張モードに、ER6データ(5F00)H、ER7データ(45)Hを書き込む
VB	Call Nmc_WriteData2(1, 1, AXIS_X, &H60, &H5F00, &H0045) ' ユニット1, IC1のX軸の拡張モードに、ER6データ(5F00)H、ER7データ(45)Hを書き込む

3．4．9．5	関数名	Nmc_ReadData	機能	命令コードにより指定のデータを読み出す

データ読み出し命令コードにより、指定軸のデータ(位置カウンタや速度など)を読み出します。

【構文】

VC	long　　Nmc_ReadData(int No, int IcNo, int Axis, int cmd);
VB	Function Nmc_ReadData(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal cmd As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	Axis	int	データを読み出す軸	入力
	cmd	int	データ読み出し命令コード((10)H～(14)H)　(下記の【注3．4．9】を参照)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	Axis	Integer	データを設定する軸	入力
	cmd	Integer	データ読み出し命令コード((10)H～(14)H)　(下記の【注3．4．9】を参照)	入力

【戻り値】

言語	型	内　容
VC	long	読み出したデータ
VB	Integer	読み出したデータ

【使用例】

VC	Data = Nmc_ReadData(2, 0, AXIS_X, 0x10); // ユニット2, IC0のX軸の論理位置カウンタを読み出す
VB	Data = Nmc_ReadData(2, 0, AXIS_X, &H10) ' ユニット2, IC0のX軸の論理位置カウンタを読み出す

【注3．4．9】データ書き込み命令コードとデータ読み出し命令コード

命令コードによる書き込み／読み出しの場合は、書き込みでは書き込む内容の設定、読み出しでは事後に読み出した内容の確認が必要です。詳細は、MCX314As/AL 取扱説明書の 5 章と 6 章を参照して下さい。

データ書き込み命令				データ読み出し命令	
コード	命　令	コード	命　令	コード	命　令
00h	レンジ設定	08h	円弧の中心点設定	10h	論理位置カウンタ読み出し
01h	加速度増加率設定	09h	論理位置カウンタ設定	11h	実位置カウンタ読み出し
02h	加速度設定	0Ah	実位置カウンタ設定	12h	現在ドライブ速度読み出し
03h	減速度設定	0Bh	COMP+レジスタ設定	13h	現在加速度/減速度読み出し
04h	初速度設定	0Ch	COMP-レジスタ設定	14h	同期バッファレジスタ読み出し
05h	ドライブ速度設定	0Dh	加速カウンタオフセット設定	－	－
06h	出力パルス数設定 単独補間ドライブ時の終点設定	0Eh	減速度増加率設定	－	－
		61h	原点検出速度設定	－	－
07h	マニュアル減速点設定	－	－	－	－

3.4.10　連続補間ドライブ関数

3．4．10．1	関数名	Nmc_2BPWriteEEPROM	機能	ユニットのメモリに2軸BP補間データを書き込む

ユニットのメモリに2軸 BP 補間データを書き込みます。

【構文】

VC	int　　Nmc_2BPWriteEEPROM(int No, int DataNo, DATA_2BP* pData2BP, int Cnt, int IpAxis);
VB	Function Nmc_2BPWriteEEPROM (ByVal No As Integer, ByVal DataNo As Ingeger, ByRef pData2BP As DATA_2BP(), ByVal Cnt As Integer ByVal IpAxis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	DataNo	int	連続補間データ番号(1～65535)	入力
	pData2BP	DATA_2BP	2軸BP補間データの構造体(ユーザー定義型)の配列の先頭アドレス	入力
	Cnt	int	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。	入力
	IpAxis	int	補間を実行する第1軸、第2軸を指定 (WR5/D0～D3の設定値と同じ値で指定する)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	DataNo	Integer	連続補間データ番号(1～65535)	入力
	pData2BP	DATA_2BP	2軸BP補間データの構造体(ユーザー定義型)の配列の先頭アドレス	入力
	Cnt	Integer	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	入力
	IpAxis	Integer	補間を実行する第1軸、第2軸を指定 (WR5/D0～D3の設定値と同じ値で指定する)	入力

【戻り値】

言語	型	内　容
VC	int	戻り値は、【注3．4．10－1】を参照
VB	Integer	戻り値は、【注3．4．10－1】を参照

【使用例】

VC

```
//-----2 軸 BP 補間データファイル･2BP_XXX.ini のオープン-----
Datafile.Open( "2BP_XXX.ini", CFile::modeRead));
//-----2 軸 BP 補間データを確保したメモリ領域に BP 補間データ数分(Cnt)を取得-----
pData2BP = new DATA_2BP[Cnt];
for (int i=0; i< Cnt; i++) {
      DATA_2BP[Cnt]にデータ取得
}
//-----取得したデータを本製品に転送-----
Status = Nmc_2BPWriteEEPROM(0, 1, pData2BP, 500, 0x04);
// ユニット No.0 に、連続補間データ No.1（BP 補間データ数 500 個）を書き込む
// 補間軸(0x04)は、第 1 軸:00(X 軸)、第 2 軸:01(Y 軸)
```

VB

```
'  -----2 軸 BP 補間データファイル･2BP_XXX.ini のオープン-----
Open "2BP_XXX.ini " For Input As #1
'  -----2 軸 BP 補間データを BP 補間データ数分(Cnt)取得-----
Dim Data2BP(Cnt-1) As DATA_2BP
For i = 0 To Cnt - 1
    Data2BP(Cnt)にデータ取得
Next
'  -----取得したデータを本製品に転送-----
Status  = Nmc_2BPWriteEEPROM(0, 1, Data2BP, 500, &H04)
'  ユニット No.0 に、連続補間データ No.1（BP 補間データ数 500 個）を書き込む
'  補間軸(&H24)は、第 1 軸:00(X 軸)、第 2 軸:01(Y 軸)
```

【注3．4．10－1： 補間データをメモリに書き込んだ結果の戻り値】

NCI_DONE	正常
NCI_NOT_INIT	未初期化
NCI_NOT_OPEN	未オープン
NCI_INV_ARG	パラメータ不正
NCI_RROC_IP	補間実行中
NCI_EEPROM_NO_MEM	メモリ空き容量不足
NCI_EEPROM_WRT_ERR	メモリ書き込みエラー
NCI_EEPROM_INV_DATA	連続補間データ不整合

3．4．10．2	関数名	Nmc_3BPWriteEEPROM	機能	ユニットのメモリに3軸BP補間データを書き込む

ユニットのメモリに3軸 BP 補間データを書き込みます。

【構文】

VC	Int　　Nmc_3BPWriteEEPROM(int No, int DataNo, DATA_3BP* pData3BP, int Cnt, int IpAxis);
VB	Function Nmc_3BPWriteEEPROM (ByVal No As Integer, ByVal DataNo As Ingeger, ByRef pData3BP As DATA_3BP(), ByVal Cnt As Integer ByVal IpAxis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	DataNo	int	連続補間データ番号(1～65535)	入力
	PData3BP	DATA_3BP	3軸BP補間データの構造体(ユーザー定義型)の配列の先頭アドレス	入力
	Cnt	int	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。	入力
	IpAxis	int	補間を実行する第1軸、第2軸、第3軸を指定 (WR5/D0～D5の設定値と同じ値で指定する)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	DataNo	Integer	連続補間データ番号(1～65535)	入力
	PData3BP	DATA_3BP	3軸BP補間データの構造体(ユーザー定義型)の配列の先頭アドレス。	入力
	Cnt	Integer	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	入力
	IpAxis	Integer	補間を実行する第1軸、第2軸、第3軸を指定 (WR5/D0～D5の設定値と同じ値で指定する)	入力

【戻り値】

言語	型	内 容
VC	int	戻り値は、【注3．4．10－1】を参照
VB	Integer	戻り値は、【注3．4．10－1】を参照

【使用例】

VC

```
//-----3 軸 BP 補間データファイル･3BP_XXX.ini のオープン-----
Datafile.Open( 3BP_XXX.ini", CFile::modeRead));

//-----3 軸 BP 補間データを確保したメモリ領域に BP 補間データ数分(Cnt)を取得-----
pData3BP = new DATA_3BP[Cnt];
for (int i=0; i< Cnt; i++) {
      DATA_3BP[Cnt]にデータ取得
}
//-----取得したデータを本製品に転送-----
Status = Nmc_3BPWriteEEPROM(0, 2, pData3BP, 400, 0x24);
// ユニット No.0 に、連続補間データ No.2（BP 補間データ数 400 個）を書き込む
// 補間軸(0x24)は、第 1 軸:00(X 軸)、第 2 軸:01(Y 軸)、第 3 軸:10(Z 軸)
```

VB

```
'  -----3 軸 BP 補間データファイル･3BP_XXX.ini のオープン-----
Open "3BP_XXX.ini " For Input As #1
'  -----3 軸 BP 補間データを BP 補間データ数分(Cnt)取得-----
Dim Data3BP(Cnt-1) As DATA_3BP
For i = 0 To Cnt - 1
    Data3BP(Cnt)にデータ取得
Next

'  -----取得したデータを本製品に転送-----
Status  = Nmc_3BPWriteEEPROM(0, 1, Data3BP, 400, &H24)
'  ユニット No.0 に、連続補間データ No.2（BP 補間データ数 400 個）を書き込む
'  補間軸(&H24)は、第 1 軸:00(X 軸)、第 2 軸:01(Y 軸)、第 3 軸:10(Z 軸)
```

3．4．10．3	関数名	Nmc_2CipWriteEEPROM	機能	ユニットのメモリに2軸連続補間データを書き込む

ユニットのメモリに2軸連続補間データを書き込みます。

【構文】

VC	int Nmc_2CipWriteEEPROM (int No, int DataNo, DATA_2CIP* pData2CIP, int Cnt, int IpAxis, BOOL SpdChgFlg);
VB	Function Nmc_2CipWriteEEPROM(ByVal No As Integer, ByVal DataNo As Integer,ByRef pData2CIP As DATA_2CIP(), ByVal Cnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	DataNo	int	連続補間データ番号(1～65535)	入力
	pData2CIP	DATA_2CIP	2軸連続補間データの構造体(ユーザー定義型)の配列の先頭アドレス	入力
	Cnt	int	2軸補間データの数。 構造体(ユーザー定義型)配列の配列数(セグメント数)を指定。	入力
	IpAxis	int	補間を実行する第1軸、第2軸を指定 (WR5/D0～D3の設定値と同じ値で指定する)	入力
	SpdChgFlg	BOOL	TRUE：補間実行中の速度変更あり、FALSE：速度変更なし	入力
VB	No	Integer	ユニット番号(0～15)	入力
	DataNo	Integer	連続補間データ番号(1～65535)	入力
	pData2CIP	DATA_2CIP	2軸連続補間データの構造体(ユーザー定義型)の配列の先頭アドレス	入力
	Cnt	Integer	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(セグメント数)を指定。	入力
	IpAxis	Integer	補間を実行する第1軸、第2軸を指定	入力
	SpdChgFlg	Integer	True：補間実行中の速度変更あり、False：速度変更なし	入力

【戻り値】

言語	型	内 容
VC	int	戻り値は、【注3．4．10－1】を参照
VB	Integer	戻り値は、【注3．4．10－1】を参照

【使用例】

VC

```
//-----2 軸連続補間データファイル･2CIP_2ST+CCW.ini のオープン-----
Datafile.Open( "2CIP_2ST+CCW.ini", CFile::modeRead));
//-----2 軸連続補間データを確保したメモリ領域にセグメント数分(Cnt)を取得-----
pData2CIP = new DATA_2CIP[Cnt];
for (int i=0; i< Cnt; i++) {
      DATA_2CIP[Cnt]にデータ取得
}
//-----取得したデータを本製品に転送-----
Status = Nmc_2CipWriteEEPROM(1, 10, pData2CIP, 8, 0x04, FALSE);
// ユニット No.1 に、連続補間データ No.10（補間セグメント 8 個）を書き込む
// 補間軸(0x04)は、第 1 軸:00(X 軸)、第 2 軸:01(Y 軸)、連続補間ドライブ中の速度変更：FALSE(なし)
```

VB

```
'  -----2 軸連続補間データファイル･2CIP_2ST+CCW.ini のオープン-----
Open "2CIP_2ST+CCW.ini " For Input As #1
'  -----2 軸連続補間データをセグメント数分(Cnt)取得-----
Dim Data2CIP(Cnt-1) As DATA_2CIP
For i = 0 To Cnt - 1
    Data2CIP(Cnt)にデータ取得
Next
'  -----取得したデータを本製品に転送-----
Status  = Nmc_2CipWriteEEPROM(1, 10, Data2CIP, 8, &H04, False)
'  ユニット No.1 に、連続補間データ No.10（補間セグメント 8 個）を書き込む
'  補間軸(&H04)は、第 1 軸:00(X 軸)、第 2 軸:01(Y 軸)、連続補間ドライブ中の速度変更：FALSE(なし)
```

3．4．10．4	関数名	Nmc_3CipWriteEEPROM	機能	ユニットのメモリに3軸連続補間データを書き込む

ユニットのメモリに3軸連続補間データを書き込みます。

【構文】

VC	int Nmc_3CipWriteEEPROM (int No, int DataNo, DATA_3CIP* pData3CIP, int Cnt, int IpAxis, BOOL SpdChgFlg):
VB	Function Nmc_3CipWriteEEPROM(ByVal No As Integer, ByVal DataNo As Integer,ByRef pData3CIP As DATA_3CIP(), ByVal Cnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	DataNo	int	連続補間データ番号(1～65535)	入力
	pData3CIP	DATA_3CIP	3軸連続補間データの構造体(ユーザー定義型)の配列の先頭アドレス	入力
	Cnt	int	3軸補間データの数。 構造体(ユーザー定義型)配列の配列数(セグメント数)を指定。	入力
	IpAxis	int	補間を実行する第1軸、第2軸、第3軸を指定 (WR5/D0～D5の設定値と同じ値で指定する)	入力
	SpdChgFlg	BOOL	TRUE：補間実行中の速度変更あり、FALSE：速度変更なし	入力
VB	No	Integer	ユニット番号(0～15)	入力
	DataNo	Integer	連続補間データ番号(1～65535)。	入力
	pData3CIP	DATA_3CIP	2軸連続補間データの構造体(ユーザー定義型)の配列の先頭アドレス。	入力
	Cnt	Integer	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(セグメント数)を指定。	入力
	IpAxis	Integer	補間を実行する第1軸、第2軸、第3軸を指定	入力
	SpdChgFlg	Integer	True：補間実行中の速度変更あり、False：速度変更なし	入力

【戻り値】

言語	型	内 容
VC	Int	戻り値は、【注3．4．10－1】を参照
VB	Integer	戻り値は、【注3．4．10－1】を参照

【使用例】

VC

```
//-----3 軸連続補間データファイル･3CIP_XXX.ini のオープン-----
Datafile.Open( "3CIP_XXX.ini", CFile::modeRead));

//-----3 軸連続補間データを確保したメモリ領域にセグメント数分(Cnt)を取得-----
pData3CIP = new DATA_3CIP[Cnt];
for (int i=0; i< Cnt; i++) {
    DATA_3CIP[Cnt]にデータ取得
}
//-----取得したデータを本製品に転送-----
Status = Nmc_3CipWriteEEPROM(1, 11, pData3CIP, 20, 0x24, TRUE);
// ユニット No.1 に、連続補間データ No.11（補間セグメント 20 個）を書き込む
// 補間軸(0x24)は、第 1 軸:00(X 軸)、第 2 軸:01(Y 軸)、第 3 軸:10(Z 軸)
// 連続補間ドライブ中の速度変更：TRUE(あり)
```

VB	' -----3 軸連続補間データファイル・3CIP_XXX.ini のオープン----- Open "3CIP_XXX.ini " For Input As #1 ' -----3 軸連続補間データをセグメント数分(Cnt)取得----- Dim Data3CIP(Cnt-1) As DATA_3CIP For i = 0 To Cnt - 1 Data3CIP(Cnt)にデータ取得 Next ' -----取得したデータを本製品に転送----- Status = Nmc_3CipWriteEEPROM(1, 11, Data3CIP, 20, &H24, TRUE) ' ユニット No.1 に、連続補間データ No.11（補間セグメント 20 個）を書き込む ' 補間軸(&H24)は、第 1 軸:00(X 軸)、第 2 軸:01(Y 軸)、第 3 軸:10(Z 軸) ' 連続補間ドライブ中の速度変更：TRUE(あり)

3．4．10．5	関数名	Nmc_2BPReadEEPROM	機能	ユニットのメモリから2軸BP補間データを読み出す

ユニットのメモリから2軸 BP 補間データを読み出します。

【構文】

VC	int Nmc_2BPReadEEPROM(int No, int DataNo, DATA_2BP* pData2BP, int BufCnt, int* DataCnt, int* IpAxis);
VB	Function Nmc_2BPReadEEPROM(ByVal No As Integer, ByVal DataNo As Integer, ByRef pData2BP As DATA_2BP, ByVal BufCnt As Integer, ByRef DataCnt As Integer, ByRef IpAxis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	DataNo	int	読み出す連続補間データ番号(1～65535)	入力
	pData2BP	DATA_2BP	読み出された2軸BP補間データの構造体の配列	出力
	BufCnt	int	読み出す2軸BP補間データ数(構造体配列の数)	入力
	DataCnt	int	読み出された2軸BP補間データ数	出力
	IpAxis	int	読み出された補間軸	出力
VB	No	Integer	ユニット番号(0～15)	入力
	DataNo	Integer	連続補間データ番号(1～65535)。	入力
	pData2BP	DATA_2BP	読み出された2軸BP補間データの構造体の配列	出力
	BufCnt	Integer	読み出す2軸BP補間データ数(構造体配列の数)	入力
	DataCnt	Integer	読み出された2軸BP補間データ数	出力
	IpAxis	Integer	読み出された補間軸	出力

【戻り値】

言語	型	内 容
VC	int	戻り値は、【注3．4．10－2】を参照
VB	Integer	戻り値は、【注3．4．10－2】を参照

【使用例】

VC

```
DATA_2BP *pData2BP = NULL;      // 読み出されたデータ

int BufCnt;                     // 読み出す2軸 BP 補間データ数
int DataCnt;                    // 読み出された2軸 BP 補間データ数
int IpAxis;                     // 読み出された補間軸

pData2BP = new DATA_2BP[BufCnt];
Status = Nmc_2BPReadEEPROM(0, 1, pData2BP, BufCnt, &DataCnt, &IpAxis);
// ユニット番号 0 の連続補間データ番号 1 の BP 補間データ数･BufCnt 分を読み出す
```

VB

```
Dim BufCnt As Long              '   読み出す2軸 BP 補間データ数
Dim DataCnt As Long             '   読み出された2軸 BP 補間データ数
Dim IpAxis As Long              '   読み出された補間軸

Dim Data2BP(BufCnt-1) As DATA_2BP   '   読み出されたデータ

For i = 0 To BufCnt - 1
    Data2BP(BufCnt)を初期化
Next

Status = Nmc_2BPReadEEPROM(0, 1, Data2BP, BufCnt, DataCnt, IpAxis)
'   ユニット番号 0 の連続補間データ番号 1 の BP 補間データ数･BufCnt 分を読み出す
```

【注3．4．10－2： 補間データをメモリから読み出した結果の戻り値】

NCI_DONE	正常
NCI_NOT_INIT	未初期化
NCI_NOT_OPEN	未オープン
NCI_INV_ARG	パラメータ不正
NCI_RROC_IP	補間実行中
NCI_EEPROM_INV_TYPE	補間処理種別不正
NCI_EEPROM_INV_DATA	連続補間データ不整合
NCI_EEPROM_NO_DATA	該当連続補間データ無し
NCI_EEPROM_NO_BUF	データ読み出しバッファ不足

3．4．10．6	関数名	Nmc_3BPReadEEPROM	機能	ユニットのメモリから3軸BP補間データを読み出す

ユニットのメモリから3軸 BP 補間データを読み出します。

【構文】

VC	int Nmc_3BPReadEEPROM(int No, int DataNo, DATA_3BP* pData3BP, int BufCnt, int* DataCnt, int* IpAxis)
VB	Function Nmc_3BPReadEEPROM(ByVal No As Integer, ByVal DataNo As Integer, ByRef pData3BP As DATA_3BP, ByVal BufCnt As Integer, ByRef DataCnt As Integer, ByRef IpAxis As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	DataNo	int	読み出す連続補間データ番号(1～65535)	入力
	PData3BP	DATA_3BP	読み出された3軸 BP 補間データの構造体の配列	出力
	BufCnt	int	読み出す3軸 BP 補間データ数(構造体配列の数)	入力
	DataCnt	int	読み出された3軸 BP 補間データ数	出力
	IpAxis	int	読み出された補間軸	出力
VB	No	Integer	ユニット番号(0～15)	入力
	DataNo	Integer	連続補間データ番号(1～65535)。	入力
	pData3BP	DATA_3BP	読み出された3軸 BP 補間データの構造体の配列	出力
	BufCnt	Integer	読み出す3軸 BP 補間データ数(構造体配列の数)	入力
	DataCnt	Integer	読み出された3軸 BP 補間データ数	出力
	IpAxis	Integer	読み出された補間軸	出力

【戻り値】

言語	型	内 容
VC	int	戻り値は、【注3．4．10－2】を参照
VB	Integer	戻り値は、【注3．4．10－2】を参照

【使用例】

VC

```
DATA_3BP *pData3BP = NULL;      // 読み出されたデータ
int BufCnt;                     // 読み出す3軸 BP 補間データ数
int DataCnt;                    // 読み出された3軸 BP 補間データ数
int IpAxis;                     // 読み出された補間軸

pData3BP = new DATA_3BP[BufCnt];
Nmc_3BPReadEEPROM(0, 2, pData3BP, BufCnt, &DataCnt, &IpAxis);
// ユニット番号 0 の連続補間データ番号 2 の BP 補間データ数･BufCnt 分を読み出す
```

VB

```
Dim BufCnt As Long              '   読み出す3軸 BP 補間データ数
Dim DataCnt As Long             '   読み出された3軸 BP 補間データ数
Dim IpAxis As Long              '   読み出された補間軸

Dim Data3BP(BufCnt-1) As DATA_3BP   '   読み出されたデータ
For i = 0 To BufCnt - 1
    Data3BP(BufCnt)を初期化
Next
Status = Nmc_3BPReadEEPROM(0, 2, Data3BP, BufCnt, DataCnt, IpAxis)
'   ユニット番号 0 の連続補間データ番号 2 の BP 補間データ数･BufCnt 分を読み出す
```

3．4．10．7	関数名	Nmc_2CiReadEEPROM	機能	ユニットのメモリから2軸連続補間データを読み出す

ユニットのメモリから2軸連続補間データを読み出します。

【構文】

VC	int Nmc_2CipReadEEPROM(int No, int DataNo, DATA_2CIP* pData2CIP, int BufCnt, int* DataCnt, int* IpAxis, BOOL* pSpdChgFlg);
VB	Function Nmc_2CIPReadEEPROM(ByVal No As Integer, ByVal DataNo As Integer, ByRef pData2CIP As DATA_2CIP, ByVal BufCnt As Integer, ByRef DataCnt As Integer, ByRef IpAxis As Integer, ByRef pSpdChgFlg As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	DataNo	int	読み出す連続補間データ番号(1～65535)	入力
	pData2CIP	DATA_2CIP	読み出された2軸連続補間データの構造体の配列	出力
	BufCnt	int	読み出す2軸補間データ数（セグメント数）	入力
	DataCnt	int	読み出された2軸補間データ数(セグメント数)	出力
	IpAxis	int	読み出された補間軸	出力
	pSpdChgFlg	BOOL	読み出された速度変更フラグ TRUE：補間実行中の速度変更あり、FALSE：速度変更なし	出力
VB	No	Integer	ユニット番号(0～15)	入力
	DataNo	Integer	読み出す連続補間データ番号(1～65535)	入力
	pData2CIP	DATA_2CIP	読み出された2軸連続補間データの構造体の配列	出力
	BufCnt	Integer	読み出す2軸補間データ数（セグメント数）	入力
	DataCnt	Integer	読み出された2軸補間データ数(セグメント数)	出力
	IpAxis	Integer	読み出された補間軸	出力
	pSpdChgFlg	Integer	読み出された速度変更フラグ	出力

【戻り値】

言語	型	内 容
VC	int	戻り値は、【注3．4．10－2】を参照
VB	Integer	戻り値は、【注3．4．10－2】を参照

【使用例】

VC

```
DATA_2CIP *pData2CIP = NULL;      // 読み出されたデータ

int BufCnt;                       // 読み出す2軸補間データ数（セグメント数）
int DataCnt;                      // 読み出された2軸補間データ数（セグメント数）
int IpAxis;                       // 読み出された補間軸
BOOL SpdChgFlg;                   // 読み出された速度変更フラグ

pData2CIP = new DATA_2CIP[BufCnt];
Status = Nmc_2CipReadEEPROM(2, 10, pData2CIP, BufCnt, &DataCnt, &IpAxis, &SpdChgFlg);
// ユニット番号 2 の連続補間データ番号 10 のセグメント数･BufCnt 分を読み出す
```

VB

```
Dim BufCnt As Long                ' 読み出す2軸補間データ数

Dim DataCnt As Long               ' 読み出された2軸補間データ数
Dim IpAxis As Long                ' 読み出された補間軸
Dim  Spdchgflg                    ' 読み出された速度変更フラグ

ReDim Data2CIP(BufCnt-1) As DATA_2BP    ' 読み出されたデータ

For i = 0 To BufCnt - 1
    Data2CIP(BufCnt)を初期化
Next
Status = Nmc_2CipReadEEPROM(2, 10, Data2CIP, BufCnt, DataCnt, IpAxis, SpdChgFlg)
' ユニット番号 2 の連続補間データ番号 10 のセグメント数･BufCnt 分を読み出す
```

3．4．10．8	関数名	Nmc_3CiReadEEPROM	機能	ユニットのメモリから3軸連続補間データを読み出す

ユニットのメモリから3軸連続補間データを読み出します。

【構文】

VC	int Nmc_3CipReadEEPROM(int No, int DataNo, DATA_3CIP* pData3CIP, int BufCnt, int* DataCnt, int* IpAxis, BOOL* pSpdChgFlg);
VB	Function Nmc_3CIPReadEEPROM(ByVal No As Integer, ByVal DataNo As Integer, ByRef pData3CIP As DATA_3CIP, ByVal BufCnt As Integer, ByRef DataCnt As Integer, ByRef IpAxis As Integer, ByRef pSpdChgFlg As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	DataNo	int	読み出す連続補間データ番号(1～65535)	入力
	pData3CIP	DATA_3CIP	読み出された3軸連続補間データの構造体の配列	出力
	BufCnt	int	読み出す3軸補間データ数(セグメント数)	入力
	DataCnt	int	読み出された3軸補間データ数(セグメント数)	出力
	IpAxis	int	読み出された補間軸	出力
	pSpdChgFlg	BOOL	読み出された速度変更フラグ	出力
VB	No	Integer	ユニット番号(0～15)	入力
	DataNo	Integer	読み出す連続補間データ番号(1～65535)	入力
	pData3CIP	DATA_3CIP	読み出された3軸連続補間データの構造体の配列	出力
	BufCnt	Integer	読み出す3軸補間データ数(セグメント数)	入力
	DataCnt	Integer	読み出された3軸補間データ数(セグメント数)	出力
	IpAxis	Integer	読み出された補間軸	出力
	pSpdChgFlg	Integer	読み出された速度変更フラグ	出力

【戻り値】

言語	型	内 容
VC	int	戻り値は、【注3．4．10－2】を参照
VB	Integer	戻り値は、【注3．4．10－2】を参照

【使用例】

VC

```
DATA_3CIP *pData3CIP = NULL;      // 読み出されたデータ

int BufCnt;                       // 読み出す3軸補間データ数(セグメント数)
int DataCnt;                      // 読み出された3軸補間データ数(セグメント数)
int IpAxis;                       // 読み出された補間軸
BOOL SpdChgFlg;                   // 読み出された速度変更フラグ

pData3CIP = new DATA_3CIP[ReadSegNum];
Nmc_3CipReadEEPROM(3, 20, pData3CIP, BufCnt, &DataCnt, &IpAxis, &SpdChgFlg);
// ユニット番号 3 の連続補間データ番号 20 のセグメント数･BufCnt 分を読み出す
```

VB

```
Dim BufCnt As Long                ' 読み出す3軸補間データ数

Dim DataCnt As Long               ' 読み出された3軸補間データ数
Dim IpAxis As Long                ' 読み出された補間軸
Dim Spdchgflg                     ' 読み出された速度変更フラグ

ReDim Data3CIP(BufCnt-1) As DATA_3BP    ' 読み出されたデータ
For i = 0 To BufCnt - 1
    Data3CIP(BufCnt)を初期化
Next
Status = Nmc_3CipReadEEPROM(3, 20, Data3CIP, BufCnt, DataCnt, IpAxis, SpdChgFlg)
' ユニット番号 3 の連続補間データ番号 20 のセグメント数･BufCnt 分を読み出す
```

3．4．10．9	関数名	Nmc_DeleteEEPROM	機能	ユニットのメモリに登録されている連続補間データを削除する

ユニットのメモリに登録されている連続補間データ(BP 補間データも含む)を削除します。

【構文】

VC	int　　Nmc_DeleteEEPROM(int No, int DataNo):
VB	Function Nmc_DeleteEEPROM(ByVal No As Inger, ByVal DataNo As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内　容	入/出
VC	No	int	ユニット番号(0～15)	入力
	DataNo	int	連続補間データ番号(1～65535、0:全データ削除)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	DataNo	Integer	連続補間データ番号(1～65535、0:全データ削除)	入力

【戻り値】

言語	型	内　容
VC	int	削除した結果 NCI_DONE 正常 / NCI_INV_ARG パラメータ不正 / NCI_NOT_INIT 未初期化 / NCI_NOT_OPEN 未オープン / NCI_RROC_IP 補間実行中 / NCI_EEPROM_INV_DATA 連続補間データ不整合 / NCI_EEPROM_NO_DATA 該当連続補間データ無し
VB	Integer	削除した結果

【使用例】

VC	Status = Nmc_DeleteEEPROM(0, 20);　　// ユニット番号0の連続補間データNo.20を削除
VB	Status = Nmc_DeleteEEPROM(0, 20)　　'　ユニット番号0の連続補間データNo.20を削除

3. 4. 10. 10	関数名	Nmc_Exec_Ip	機能	連続補間ドライブ処理を実行する

連続補間ドライブ(BP 補間ドライブも含む)処理を実行します。

【構文】

VC	int Nmc_Exec_Ip (int No, int IcNo, int DataNo);
VB	Function Nmc_Exec_Ip(ByVal No As Integer, ByVal IcNo As Integer, ByVal DataNo As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	DataNo	int	連続補間データ番号	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	DataNo	Integer	連続補間データ番号	入力

【戻り値】

言語	型	内 容
VC	int	実行開始の結果
VB	Integer	実行開始の結果

NCI_DONE	正常		
NCI_INV_ARG	パラメータ不正	NCI_RROC_IP	補間実行中
NCI_NOT_INIT	未初期化	NCI_EEPROM_INV_DATA	連続補間データ不整合
NCI_NOT_OPEN	未オープン	NCI_EEPROM_NO_DATA	該当連続補間データ無し

【使用例】

VC

```
//-----補間モードレジスタ(WR5)設定:ユニット番号3のIC0に設定―――
Nmc_WriteReg5(3, 0, 0x4104);                //連続補間割込み許可(CIINT) 、線速一定、1軸:X、2軸:Y

//-----パラメータ設定-----
Nmc_Range(3, 0, AXIS_X, 4000000);     // レンジ設定(X軸)          (倍率2)
Nmc_Range(3, 0, AXIS_Y, 5656000);     // レンジ設定(Y軸)          (2軸線速一定のためのレンジ)
Nmc_StartSpd(3, 0, AXIS_X, 500);      // 初速度(SV)設定           (500×2 = 1000PPS)
Nmc_Speed(3, 0, AXIS_X, 500);         // ドライブ速度(V)設定      (500×2 = 1000PPS)
Nmc_Exec_Ip(3, 0, 20);                // 連続補間ドライブNo.20を実行する
```

VB

```
'  -----補間モードレジスタ(WR5)設定:ユニット番号3のIC0に設定―――
Call Nmc_WriteReg5(3, 0, 0x4104)          '   連続補間割込み許可(CIINT) 、線速一定、1軸:X、2軸:Y

'  -----パラメータ設定-----
Call Nmc_Range(3, 0, AXIS_X, 4000000)     '  レンジ設定(X軸)          (倍率2)
Call Nmc_Range(3, 0, AXIS_Y, 5656000)     '  レンジ設定(Y軸)          (2軸線速一定のためのレンジ)
Call Nmc_StartSpd(3, 0, AXIS_X, 500)      '  初速度(SV)設定           (500×2 = 1000PPS)
Call Nmc_Speed(3, 0, AXIS_X, 500)         '  ドライブ速度(V)設定      (500×2 = 1000PPS)
Status = Nmc_Exec_Ip(3, 0, 20)            '  連続補間ドライブNo.20を実行する
```

3. 4. 10. 11	関数名	Nmc_Stop_Ip	機能	連続補間ドライブ処理を停止する

連続補間ドライブ(BP 補間ドライブも含む)処理を停止します。連続補間ドライブを途中で停止する場合は、この関数を使用して下さい。

【構文】

VC	int Nmc_Stop_Ip(int No, int IcNo);
VB	Function Nmc_Stop_Ip(ByVal No As Inger, ByVal IcNo As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力

【戻り値】

言語	型	内 容
VC	int	停止した結果 NCI_DONE 正常 NCI_NOT_INIT 未初期化 NCI_INV_ARG パラメータ不正 NCI_NOT_OPEN 未オープン NCI_NOT_START 補間未実行
VB	Integer	停止した結果

【使用例】

VC	Status = Nmc_Stop_Ip(5, 1); // ユニット番号5のIC1(増設4軸基板)の連続補間ドライブを停止
VB	Status = Nmc_Stop_Ip(5, 1) ' ユニット番号5のIC1(増設4軸基板)の連続補間ドライブを停止

3. 4. 10. 12	関数名	Nmc_StsRead_Ip	機能	連続補間ドライブ実行状態を取得する

連続補間ドライブ(BP 補間ドライブも含む)の実行状態を取得します。

【構文】

VC	int Nmc_StsRead_Ip(int No, int IcNo, IP_EXEC_STATUS* pStatus);
VB	Function Nmc_StsRead_Ip (ByVal No As Integer, ByVal IcNo As Integer, ByRef pStatsus As IP_EXEC_STATUS) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC番号(0,1)	入力
	pStatus	IP_EXEC_STATUS	取得した情報が格納される補間実行状態情報構造体 (ﾕﾆｯﾄNo,ICNo,実行状態,補間種別,ｾｸﾞﾒﾝﾄNo.,補間軸,主軸の論理/実位置ｶｳﾝﾀ,主軸の現在ﾄﾞﾗｲﾌﾞ速度)	出力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	pStatsus	IP_EXEC_STATUS	取得した情報が格納される補間実行状態情報構造体	出力

【戻り値】

言語	型	内 容
VC	int	取得した結果 NCI_DONE 正常 NCI_NOT_INIT 未初期化 NCI_INV_ARG パラメータ不正 NCI_NOT_OPEN 未オープン
VB	Integer	取得した結果

【使用例】

VC	IP_EXEC_STATUS IpBuf; //取得した情報が格納されるエリアの先頭アドレス
	Status = Nmc_StsRead_Ip(5, 1, &IpBuf); //ユニット5/IC1の連続補間ドライブ実行状態を取得します

VB	Dim IpBuf As IP_EXEC_STATUS ' 取得した情報が格納されるエリアの先頭アドレス Status = Nmc_StsRead_Ip(5, 1, IpBuf) ' ユニット5/IC1の連続補間ドライブ実行状態を取得します

3.4.11　アプリケーションから制御する補間ドライブ関数

3．4．11．1	関数名	Nmc_Command_IP	機能	補間命令を実行する

指定 IC の WR0 レジスタに、補間命令番号を書き込みます。
補間軸の設定は、Nmc_WriteReg5 で行います。

【構文】

VC	void　Nmc_Command_IP(int No, int IcNo, int cmd);
VB	Sub　Nmc_Command_IP(ByVal No As Integer, ByVal IcNo As Integer, ByVal cmd As Integer)

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC 番号(0,1)	入力
	cmd	int	補間命令番号　（3．4節（6）の「補間ドライブ命令」を設定）	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC番号(0,1)	入力
	cmd	Integer	補間命令番号	入力

【戻り値】 なし

【使用例】

VC	Nmc_WriteReg5(0, 0, 0x0004);	// ユニット番号 0/IC0 の補間軸設定（主軸：X、第2軸：Y）
	Status = Nmc_Command_IP(0, 0, CMD_IP_2ST);	// 2軸直線補間実行
VB	Call Nmc_WriteReg5(0, 0, 0x0004)	' ユニット番号 0/IC0 の補間軸設定（主軸：X、第2軸：Y）
	Status = Nmc_Command_IP(0, 0, CMD_IP_2ST)	' 2軸直線補間実行

3．4．11．2	関数名	Nmc_GetCNextStatus	機能	次の補間データ書き込み状態を取得する

アプリケーションから制御する連続補間ドライブにおいて、次の補間データの書き込みが可能か否かの状態を取得します。

【構文】

VC	int　　Nmc_GetCNextStatus(int No, int IcNo);
VB	Function Nmc_GetCNextStatus(ByVal No As Integer, ByVal IcNo As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC 番号(0,1)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC 番号(0,1)	入力

【戻り値】

言語	型	内 容
VC	int	次補間データ書込み可能の場合は、0以外を返す。 次補間データ書込み可能ではない場合は、0を返す。
VB	Integer	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 (2, 0) != 0)
	　　AfxMessageBox(“ユニット番号2/IC0/次の補間データの書き込みが可能である”);
	else
	　　AfxMessageBox(“ユニット番号 2/IC0/次の補間データの書き込みが可能でない”);

VB	If Nmc_ GetCNextStatus (2, 0) <> 0 Then
	　　Call MsgBox(“ユニット番号2/IC0/次の補間データの書き込みが可能である”)
	Else
	　　Call MsgBox(“ユニット番号2/IC0/次の補間データの書き込みが可能でない”)
	End If

3．4．11．3	関数名	Nmc_GetBpSc	機能	BP補間スタックカウンタ値を取得する

アプリケーションから制御するビットパターン補間ドライブを連続して行う際、現在のビットパターン補間スタックカウンタ値を取得します。

【構文】

VC	int　　Nmc_GetBpSc(int No, int IcNo);
VB	Function Nmc_GetBpSc(ByVal No As Integer, ByVal IcNo As Integer) As Integer

【入力パラメータ】

言語	パラメータ	型	内 容	入/出
VC	No	int	ユニット番号(0～15)	入力
	IcNo	int	IC 番号(0,1)	入力
VB	No	Integer	ユニット番号(0～15)	入力
	IcNo	Integer	IC 番号(0,1)	入力

【戻り値】

言語	型	内 容
VC	int	現在のビットパターン補間スタックカウンタの値
VB	Integer	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(2, 1);　　// ユニット番号 2/IC0/BP 補間スタックカウンタ値を取得
VB	Data = Nmc_GetBpSc(2, 1)　　' ユニット番号 2/IC0/BP 補間スタックカウンタ値を取得

4. 使用方法

4.1 制御の開始と終了

本製品の制御の開始処理と終了処理について説明します。
尚、1台の PC に対して、実行可能なアプリケーションは 1 個ですので、ご注意ください。

（1） 通信デバイスの指定

制御アプリケーションの処理を開始する際は、本製品と PC を接続する通信方式（LAN または USB）を指定して、Nmc_Initialize ( )関数で DLL を初期化し、処理の終了の際は、Nmc_Finalize ( )関数で DLL を終了します。
1つの制御アプリケーションで、LAN 接続と USB 接続の両方を選択することはできません。
本製品専用 API と DLL モジュールが、通信処理を行いますので、制御アプリケーションは、通信に関わる処理を意識することなく、本製品を制御する事ができます。

（2） 本製品（ユニット）のオープンとクローズ

通信デバイスの指定（DLL 初期化）の後は、制御するユニットのユニット番号（0～15）を指定して、Nmc_Open( )関数でユニットのオープンを行います。制御アプリケーションで割り込み処理が必要な場合は、Nmc_Open( )関数で、“割り込み処理を使用するフラグ”を TRUE（使用する）に設定します。

ユニット番号は、ロータリスイッチで設定（0～F）されます。複数台を制御する場合は、ユニット番号が重複しないようにご注意ください。
ユニットの制御を終了する際は、Nmc_Close( )関数、又は Nmc_CloseAll( )関数でユニットをクローズして下さい。
割り込み処理を行う場合は、割り込みユーザー関数（Nmc_SetEvent で指定した関数）を実行中にクローズ処理を実行しないで下さい。クローズ処理を行う場合は、必ず、割り込みユーザー関数が終了している状態で行って下さい。

（3） モータコントロール IC（MCX314AL）の初期設定

モータの駆動システムに合わせて、モータコントロール IC（MCX314AL）のパラメータ類を初期設定する必要があります。MR540 の場合は、メイン基板の MCX314AL に対して行い、MR580 の場合は、メイン基板と増設 4 軸基板の両方の MCX314AL に対して設定する必要があります。主な内容は次の通りです。

① ソフトリセット Nmc_Reset( )

② 駆動システム仕様に合わせたモード設定 Nmc_WriteReg1( )、Nmc_WriteReg2( )、Nmc_WriteReg3( )
- ・直線加減速／S 字加減速
- ・オーバランリミット入力信号の論理レベル設定
- ・サーボモータ用入出力信号、偏差カウンタ出力
- ・エンコーダ入力信号などの機能の有効／無効、論理レベル設定
- ・自動原点出し
- ・入力信号フィルタ
- ・汎用入力信号／汎用出力信号

③ 各軸の動作パラメータを設定

・レンジ	・ドライブ速度	・COMP＋レジスタ値
・加速度増加率／減速度増加率	・出力パルス数	・COMP－レジスタ値
・加速度／減速度	・論理位置カウンタ値	・加速カウンタオフセット値
・初速度	・実位置カウンタ値	・原点検出速度の設定

4.2 入出力信号のモード設定

各種入出力信号を、システムに必要な機能として使用するには、モード設定が必要です。モード設定は、ライトシステム関数を使用して、MCX314AL のライトレジスタに設定します。この節では、モード設定の概要について説明します。MCX314AL のライトレジスタの詳細内容については、MCX314As/AL 取扱説明書を参照して下さい。

入出力信号・機能	主な API 関数 ＊1	設定概要・機能	MCX314AsAL 取扱説明書
ドライブパルス 出力	Nmc_WriteReg2()	ドライブパルスの出力方式（独立 2 ﾊﾟﾙｽ／1 ﾊﾟﾙｽ）やドライブパルスの論理レベルを設定できます。	2.9.2 節 4.5 節
オーバランリミット 入力信号	Nmc_WriteReg2()	＋方向リミット／－方向リミット入力信号のそれぞれの論理レベル（Low/Hi ｱｸﾃｨﾌﾞ）とドライブ停止方式（即停止／減速停止）を設定します。	4.5 節
自動原点出し	Nmc_WriteReg1() Nmc_ExpMode(), 他	自動原点出し用信号とドライブ停止用入力信号は、兼用していますので、両方の機能を同時に使用できません。 自動原点出しの設定は、本取扱説明書の 4.4 節を参照して下さい。	2.5 節
ドライブ停止 入力信号	Nmc_WriteReg1()	ドライブ停止入力信号として使用する場合は、ドライブ停止入力信号の有効設定を行い、入力信号の論理レベル（Low で停止／Hi で停止）を設定します。	4.4 節
サーボモータ用 入力信号	Nmc_WriteReg2()	サーボモータアラーム用入力信号と位置決め完了用入力信号の有効設定を行い、論理レベル（Low ｱｸﾃｨﾌﾞ／Hi ｱｸﾃｨﾌﾞ）を設定します。	4.5 節
サーボモータ用 出力信号	Nmc_ExpMode()	サーボモータ用出力信号の偏差カウンタクリア信号として使用する場合は、信号の有効設定を行い、論理レベル（Low/Hi ｱｸﾃｨﾌﾞ）を設定し、出力信号のパルス幅を設定します。	2.5.3 節 6.16 節
汎用出力信号	Nmc_WriteReg4()	MCX314AL の WR4 アウトプットレジスタの対応のビットに 0/1 を書き込むことにより、汎用出力が制御されます。	2.9.8 節 4.7 節
汎用入力信号	Nmc_WriteReg1() Nmc_WriteReg2()	自動原点出し、ドライブ停止入力信号、サーボモータ用入力信号の機能において使用しない入力信号を、汎用入力信号として使用できます。、汎用入力信号の読み出し方法は、本取扱説明書の 4.3 節を参照して下さい。	4.14 節
エンコーダ 入力信号	Nmc_WriteReg2()	エンコーダ入力方式（2 相ﾊﾟﾙｽ入力／ｱｯﾌﾟﾀﾞｳﾝ入力）と2相パルス入力の場合は、分周比を設定します。	4.5 節

＊1 他の設定用 API 関数:
- WR1,WR2 レジスタへの書き込みには、Nmc_WriteReg()や Nmc_WriteRegSetAxis()関数も使用が可能です。
- WR4レジスタへの書き込みには、Nmc_WriteReg()関数も使用が可能です。
- 自動原点出しやサーボモータ用出力信号の拡張モード設定には、Nmc_WriteData2()関数も使用が可能です。

4.3 装置の状態確認

本製品の状態確認は、MCX314AL のリードレジスタを読み出すことにより行うか、データ読み出し関数や状態取得関数で、行います。この節では、基本的な確認方法について説明します。
MCX314AL のリードレジスタの詳細内容については、MCX314As/AL 取扱説明書を参照して下さい。

状態確認	主な API 関数 ＊1	内容概要・機能	MCX314AsAL 取扱説明書
ドライブ状態	Nmc_ReadReg0()	各軸のドライブ状態（ﾄﾞﾗｲﾌﾞ中／停止）やエラー発生中であるかが分かります。	4.10 節 4.11 節
	Nmc_ReadReg1()	指定軸のドライブ中詳細（加速/定速/減速状態）やドライブを終了した要因などが分かります。	
	Nmc_GetDriveStatus()	指定軸が、ドライブ中であるかいなかが分かります。	
エラー状態	Nmc_ReadReg2()	現在のエラー状態が分かります。（ｿﾌﾄﾘﾐｯﾄ+/−、ﾊｰﾄﾞﾘﾐｯﾄ+/−、サーﾎﾞﾓｰﾀ用ｱﾗｰﾑ、EMG 信号、自動原点出し実行時のｴﾗｰ）	4.12 節
位置カウンタ	Nmc_ReadLp()	論理位置カウンタを読み出します。	2.3.1 節
	Nmc_ReadEp()	実位置カウンタを読み出します。	7 節
現在速度	Nmc_ReadSpeed()	現在のドライブ速度を読み出します。	7 節
	Nmc_ReadAccDec()	現在の加速度、減速度を読み出します。	
汎用入力状態	Nmc_ReadReg4() Nmc_ReadReg5()	各軸の nHOME, nECZ, nALARM, nINPOS 入力状態を確認できます。従って、入力信号の機能を使用しないときは、汎用入力信号として使用する事ができます。 入力信号端子状態とレジスタ表示の関係は、次の通りです。 ・入力信号端子がハード的にオープン状態：信号対応ビットは'1' ・入力信号端子が GND と短絡状態　　：信号対応ビットは'0' をそれぞれ示します。	4.14 節
割り込み発生要因	Nmc_ReadEvent()	割り込み発生要因を読み出します。 本取扱説明書の 4.5 節を参照して下さい。	2.5～2.7 節 4.13 節
連続補間ドライブ実行状態	Nmc_StsRead_Ip()	連続補間ドライブ（BP 補間ドライブも含む）の実行状態を読み出します。（取得情報：補間実行/停止、補間種別、セグメント番号等）	——

ドライブ状態読み出しや 1 軸の位置カウンタ読み出しなど 1 つの API 関数の処理時間は、通信処理も加わりますので、約 1msec がかかります。仮に、状態監視処理が 1 軸当たり 5 項目を読み出し、4 軸分を処理すると状態監視処理だけで 20msec の時間がかかります。
従って、状態監視処理においては、必要最低限の項目とする事をお奨めします。

＊1 他の状態確認用 API 関数：
・RR0 レジスタの読み出しには、Nmc_ReadReg()関数も使用が可能です。
・RR1,RR2 レジスタの読み出しには、Nmc_ReadReg()や Nmc_ReadRegSetAxis()関数も使用が可能です。
・位置カウンタや現在速度の読み出しには、Nmc_ReadData()関数も使用が可能です。

4.4 自動原点出し

自動原点出し処理の設定項目と使用する API 関数について説明します。
MR540 の場合は、メイン基板の MCX314AL に対して設定を行い、MR580 の場合は、メイン基板と増設 4 軸基板の両方の MCX314AL に対して設定を行う必要があります。
MCX314AL の自動原点出し機能の詳細は、MCX314As/AL 取扱説明書の 2.5 節を参照して下さい。

（1）1 つの原点信号で高速原点出しを行う方法

自動原点出しに使用される信号の回路を次に示します。MCX314AL のnIN0 と nIN1 端子が、また nIN2 と nIN3 端子が短絡されていますので、1つの原点信号で高速原点出しを行う自動原点出しとなります。

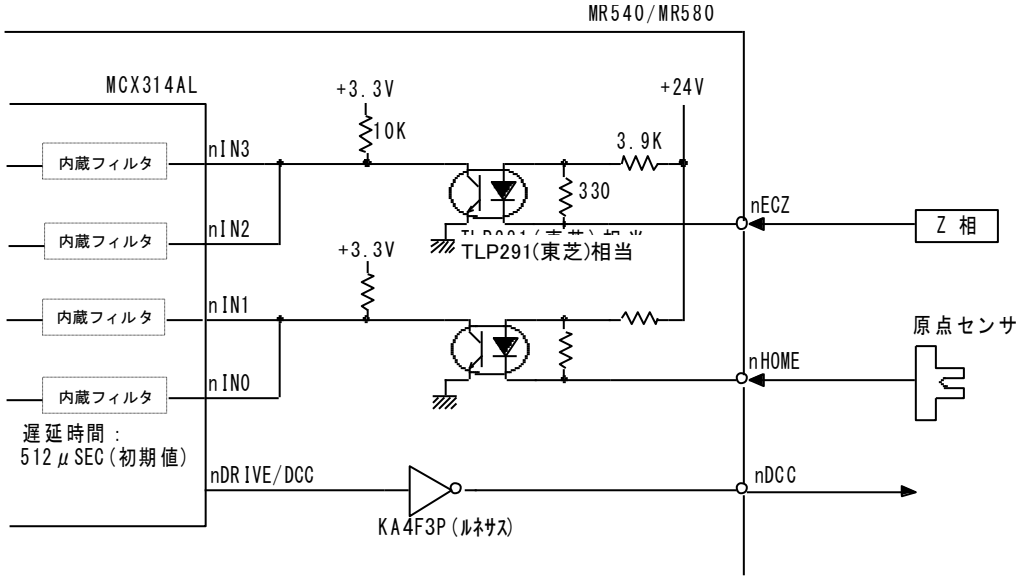


図 4.4-1　自動原点出しに関連する信号回路（1）

この場合の自動原点出しの設定項目は、次の通りです。
駆動システムで必要とする原点出し動作に合わせて、設定を行ってください。MCX314AL の nIN0 と nIN1 端子は回路上で短絡されていますので、ステップ1とステップ2の論理レベルと検出方向は、同じに設定する必要があります。

自動原点出し機能フェーズ	動　作（MCX314AL の信号）	設定内容			
		実行	論理レベル	検出方向	他
ステップ1	高速原点検出(nIN0)	有／無	Low/Hi ｱｸﾃｨﾌﾞ	＋／－方向	ドライブ速度
ステップ2	低速原点検出(nIN1)	有／無	同論理に設定	同方向に設定	原点検出速度
ステップ3	ｴﾝｺｰﾀﾞ Z 相検出(nIN2)	有／無	Low/Hi ｱｸﾃｨﾌﾞ	＋方向／－方向	原点検出速度
ステップ4	高速オフセット移動	有／無	－	＋方向／－方向	ドライブ速度
偏差ｶｳﾝﾀｸﾘｱ出力	DCC 信号出力(nDRIVE/DCC)	有／無	Low/Hi ｱｸﾃｨﾌﾞ	－	ｱｸﾃｨﾌﾞﾊﾟﾙｽ幅

設定内容は次の通りです。実際に、ご使用になる機能につきまして設定を行ってください。

API	入力パラメータ	設定概要
Nmc_WriteReg1( )	wdata/D7,D5,D3,D1=0	MCX314AL/nIN3～nIN0 信号のドライブ停止入力信号を無効とする。
	wdata/D6,D4,D2,D0=X	MCX314AL/nIN3～nIN0 信号の論理レベルを設定する。D6=D4,D2=D0 とする。
Nmc_ExpMode( )	EM6/D5=X	自動原点出し終了の割り込みを設定する。
	EM7/D7---D0＝X	ステップ 1～ステップ 4 の実行の有無と移動の方向を設定する。
	EM7/D9,D8=X,X	ステップ3･Z 相信号の処理と、ステップ4終了後の位置カウンタ値の処理を設定する。
	EM7/D15--D11=X	DCC 信号出力に関する設定を行う。
Nmc_Speed( )	wdata/1～8,000	ステップ 1、ステップ 4 の高速サーチ･ドライブ速度を設定する。
Nmc_HomeSpd( )	wdata/1～8,000	ステップ 2、ステップ 3 の低速サーチ･ドライブ速度を設定する。
Nmc_Pulse( )	wdata/0-4,294,967,295	ステップ 4 使用時のオフセット移動パルス数を設定する。

（２）リミット信号のみで原点出しを行う方法

片方のリミット信号を原点信号として原点出しを行うことができます。この方法では、原点とする片方のリミット信号とnHOME 入力端子を接続して使用します。この場合、次の2項目が条件となります。

条件①　高速で原点検出(リミット検出)動作を行う場合は、
リミット信号がアクティブになる位置から機械的な
リミットまでの距離内で充分に減速停止できる事。

条件②　自動原点出しを開始する位置が、検出方向に
向かって、リミット信号アクティブ区間を越えた先にない事。

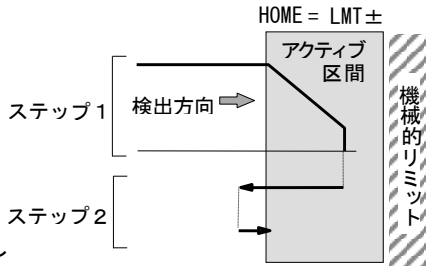


図 4.4-2　リミット信号による自動原点出し

自動原点出し以外のドライブにおいては、原点信号としたリミット信号がアクティブ状態となると、オーバリミット信号として動作します。接続方法は、原点信号とするリミット信号（nLMT+/－）とnHOME 信号を短絡します。

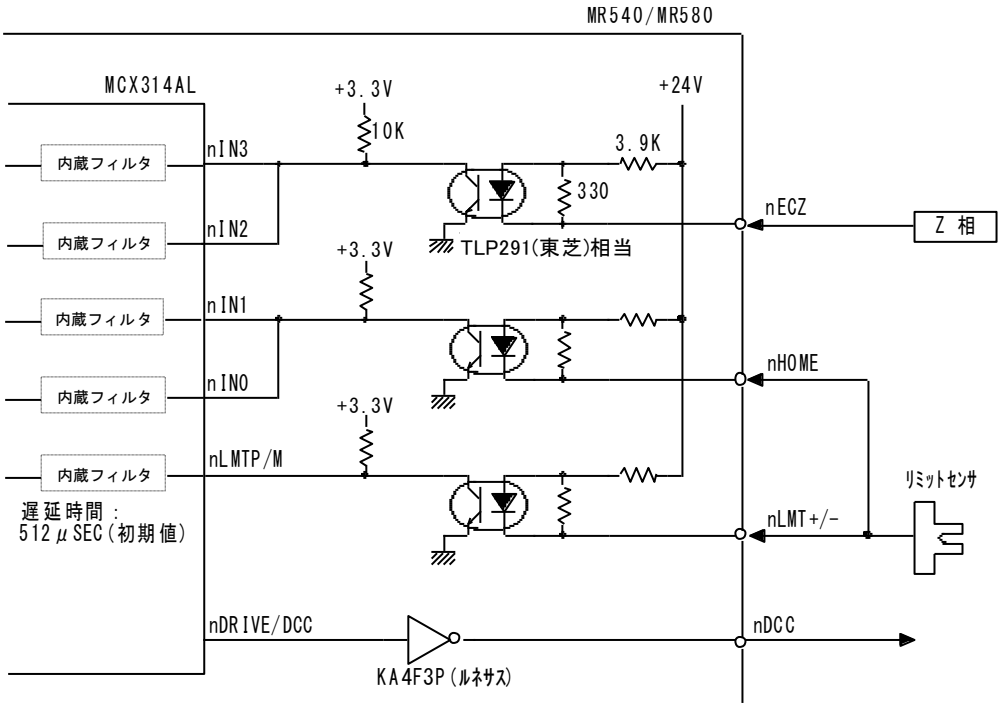


図 4.4-3　自動原点出しに関連する信号回路と接続方法

設定内容は次の通りです。実際に、ご使用になる機能につきまして設定を行ってください。

API	入力パラメータ	設定概要
Nmc_WriteReg1( )	wdata/D7,D5,D3,D1=0 wdata/D6,D4,D2,D0=X	MCX314AL/nIN3～nIN0 信号のドライブ停止入力信号を無効とする。 MCX314AL/nIN3～nIN0 信号の論理レベルを設定する。D6=D4,D2=D0 とする。
Nmc_WriteReg2( )	wdata/D2=1	リミット停止モードは、減速停止モードを設定する。
Nmc_ExpMode( )	EM6/D5=X EM7/D7---D0＝X EM7/D9,D8=X,X EM7/D10=1 EM7/D15--D11=X	自動原点出し終了の割り込みを設定する。 ステップ 1～ステップ 4 の実行の有無と移動の方向を設定する。 ステップ3･Z 相信号の処理と、ステップ4終了後の位置カウンタ値の処理を設定する。 オーバランリミット信号を使用した自動原点出しモードを設定する。 DCC 信号出力に関する設定を行う。
Nmc_Speed( )	wdata/1～8,000	ステップ 1、ステップ 4 の高速サーチ･ドライブ速度を設定する。
Nmc_HomeSpd( )	wdata/1～8,000	ステップ 2、ステップ 3 の低速サーチ･ドライブ速度を設定する。
Nmc_Pulse( )	wdata/0-4,294,967,295	ステップ 4 使用時のオフセット移動パルス数を設定する。

4.5 割り込み

割り込みの種別は多数ありますが、個別に有効／無効(許可／禁止)の設定ができます。
ただし、割り込み信号は 1 本ですので、割り込みが発生した場合にはその発生要因を確認する必要があります。
MR580 の場合は、メイン基板と増設 4 軸基板のそれぞれから割り込みが発生します。
連続補間ドライブのようなドライブ性能を重視する場合は、割り込みの使用を最小限にして下さい。

(1)機能概要

Nmc_WriteReg1( )で設定する割り込み要因は、次の通りです。

① 1つのドライブパルスを出力した。
② 論理／実位置カウンタがCOMP-レジスタ値を越えて大きくなった。
③ 論理／実位置カウンタがCOMP-レジスタ値を越えて小さくなった。
④ 論理／実位置カウンタがCOMP+レジスタ値を越えて小さくなった。
⑤ 論理／実位置カウンタがCOMP+レジスタ値を越えて大きくなった。
⑥ 加減速ドライブで、定速域でのパルス出力を終了した。
⑦ 加減速ドライブで、定速域でのパルス出力を開始した。
⑧ ドライブが終了した。

自動原点出し終了割り込みは、拡張モード設定関数 Nmc_ExpMode( )で設定します。
同期動作による割り込みは、同期動作モード設定関数 Nmc_SyncMode( )で設定します。
アプリケーションから制御する単独の補間ドライブ(Nmc_Command_IP()で行う補間ドライブ)において、補間割り込み(CIINT)とビットパターン補間割り込み(BPINT)を使用する場合は、Nmc_WriteReg5( )で設定します。
連続補間ドライブ(ファームウェアが実行する補間ドライブ)での CIINT と BPINT の割り込みは、アプリケーション側には発生しませんので、ご注意ください。
連続補間ドライブ軸への Nmc_WriteReg1( )による割り込みは、第 1 軸(主軸)のみ設定可能(第 1 軸のみ有効)です。

割り込みが発生した場合は、その割り込み要因を確認する必要があります。割り込み処理を行うユーザー関数の中で、Nmc_ReadEvent( )を使用して割り込み要因を確認して下さい。MCX314AL の割り込み機能の詳細については、MCX314As/AL 取扱説明書 2.5.3 節、2.6 節、2.7 節、4.4 節、4.13 節を参照して下さい。

(2) 割り込み処理方法

割り込み処理に関連する API 関数は次の通りです。

API	処理概要
Nmc_Open( )	オープン時に割り込み処理の使用有無を設定する。
Nmc_WriteReg1( )	使用する割り込みの許可/禁止を設定する。
Nmc_SetEvent( )	割り込み処理のユーザー関数を登録する。
Nmc_ResetEvent( )	割り込み処理のユーザー関数設定を解除する。
Nmc_ReadEvent( )	割り込みが発生して、登録済みのユーザー関数が呼び出された後、割り込み要因を確認する。

【使用例】

```
Nmc_Open 関数にて割り込み使用を設定      : Nmc_Open(No, TRUE);     // TRUE : 割り込みを使用する
Nmc_SetEvent 関数にてユーザー関数を設定: Nmc_SetEvent(No, IcNo, MC_EventProc, lpParam);
使用する(許可する)割り込みを設定          : Nmc_WriteReg1(No, IcNo, AXIS_ALL, 0x8000); // 停止時割り込み
```

割り込みが発生すると、Nmc_SetEvent 関数で設定した割り込みユーザー関数が呼び出されます。
割り込みユーザー関数では、Nmc_ReadEvent 関数にて割り込み要因を確認します。
割り込み処理するユーザー関数の設定を解除する場合は Nmc_ResetEvent 関数を実行して下さい。
この関数を実行すると、ユニットで割り込みが発生してもユーザー関数は呼び出されません。

(3) 割り込み要因のクリア

Nmc_ReadEvent()関数では、それまでに発生した割り込み要因が OR 条件で取得されます。関数を呼び出した後は、その内容がクリアされます。以前に発生した割り込み要因は、確認できませんのでご注意ください。

4.6 連続補間ドライブ

連続補間ドライブ(ビットパターン補間ドライブも含む)は、複数の補間ドライブを停止せずに連続して実行する機能です。連続補間データは、PC から本製品に転送して、ファームウェアが連続補間ドライブを実行しますので、PC の OS や PC と本製品の通信処理に影響されずに、スムーズな動作が可能です。

(1)機能概要

連続補間ドライブを実行は、次の手順で行います。

[処理1]連続補間データの作成

連続補間ドライブの内容を記載した連続補間データ(テキストファイル、INI ファイル形式)を作成します。
連続補間データの作成では、補間ドライブ種別(2/3 軸直線補間／円弧補間／2/3 軸 BP 補間／)と終点、円弧補間の場合は中心点を記述します。また、速度変更をする場合は、速度パラメータを記述します。

[処理2]連続補間データの書き込み

連続補間データ番号を付与して、本製品のフラッシュメモリに書き込みます。

[処理3]連続補間ドライブのモード設定

補間軸の指定、補間速度設定、割り込み設定等を行います。

[処理4]連続補間ドライブ実行

連続補間データ番号を指定して、実行命令を発行すると、ファームウェア制御による連続補間ドライブが実行されます。

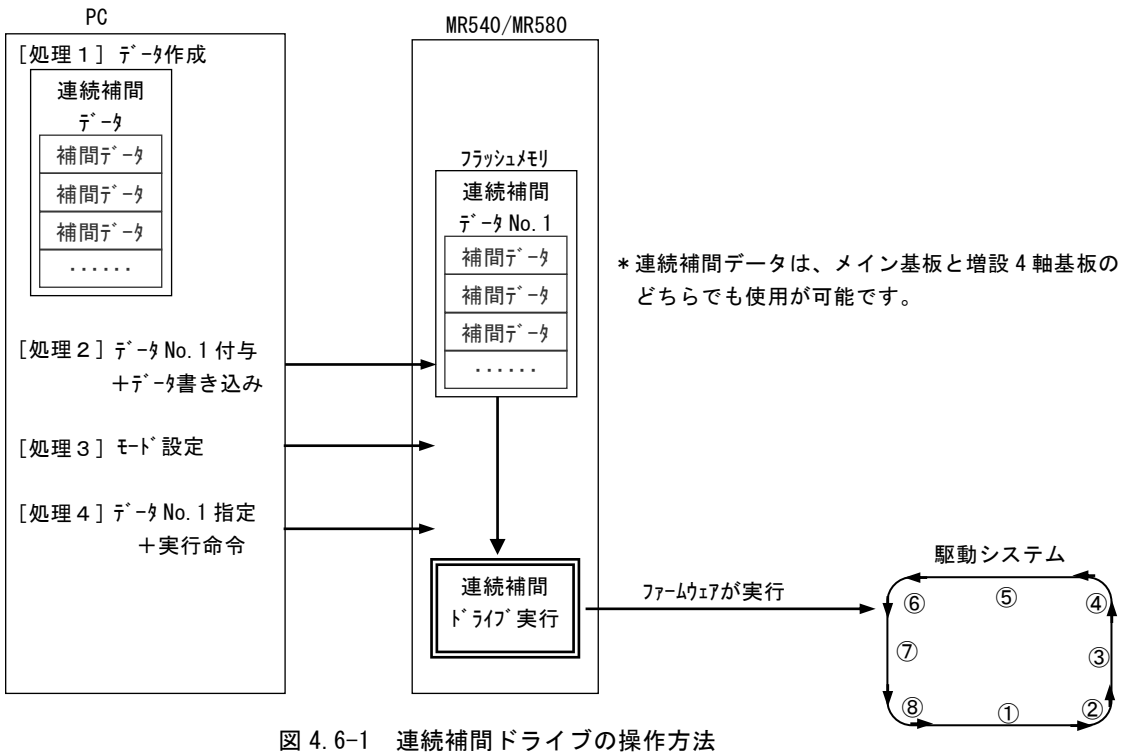


図 4.6-1　連続補間ドライブの操作方法

連続補間ドライブの制限事項と注意事項は、次の通りです。

・MR580 の場合は、メイン基板と増設 4 軸基板のそれぞれの基板内の軸間で補間ドライブを行います。
異なるユニットの軸間では、補間ドライブができませんので、ご注意ください。
・MR580 の場合は、メイン基板と増設 4 軸基板において、同時に 2 個の連続補間ドライブはできません。
・連続補間ドライブ(直線／円弧補間)の 1 セグメントの動作時間は、1msec 以上として下さい。
・BP 補間ドライブのドライブ速度は、MAX50kpps とします。
・連続補間ドライブの停止は、Nmc_Stop_Ip()関数を使用して下さい。
・連続補間ドライブ中は、アプリケーションから制御する補間ドライブ(4.7 節)は、行わないでください。
・連続補間ドライブ中のドライブ速度などの読み出しは、個々の関数で読み出すのではなく、補間実行状取得関数、Nmc_StsRead_Ip()を使用して下さい。
・MR540 を使用する際、関数の IC 番号の引数には、'0'を必ず指定して下さい。

（２）[処理１]連続補間データの作成

連続補間ドライブの内容を、連続補間データファイル(テキストファイル、INI ファイル形式)に記載します。
連続補間データファイルの 1 行目は、ヘッダの"[IP_DATA]"とし、2 行目以降の補間データは、1セグメント分のデータを 1 行に記述します。パラメータ間は','カンマ区切りとし、行の終了は[CR](Carriage Return)とします。
コメントを記述する場合は、行の先頭に';'を記述します。

a.2 軸連続補間ドライブ用データ構成

```
[IP_DATA] [CR]
  ;2CIP_No=2 軸直線補間/円弧補間,速度,終点 1,終点 2,中心 1,中心 2[CR]
   2CIP_0001=Command,Speed,EndP1,EndP2,Center1,Center2[CR]   ← 補間セグメント(補間ﾄﾞﾗｲﾌﾞ)
   2CIP_0002=Command,Speed,EndP1,EndP2,Center1,Center2[CR]   ←
    ·····  , ··· ,··· , ···, ··· ,  ···  [CR]                ←
```

データ	名 称	内 容
[IP_DATA]	セクション	データのヘッダ、先頭に記述
2CIP_nnnn=	2 軸連続補間キー	nnnn : セグメント番号(0～9999) 上昇のシリアル番号
Command	補間ドライブ種別	CMD_IP_2ST : 2 軸直線補間 CMD_IP_CW : CW 円弧補間 CMD_IP_CCW : CCW 円弧補間
Speed	速度パラメータ	補間セグメントの速度指定 0～8000、0:前セグメントと同じ速度 (先頭のセグメント(1 セグメント目)の速度は、連続補間ドライブ開始前のモード設定で行います。(4)説を参照。制限事項は(6)節を参照)
EndP1	終点(第 1 軸)	-2,147,483,646～+2,147,483,646
EndP2	終点(第 2 軸)	-2,147,483,646～+2,147,483,646
Center1	円弧中心点(第 1 軸)	-2,147,483,646～+2,147,483,646 (円弧補間のみ設定)
Center2	円弧中心点(第 2 軸)	-2,147,483,646～+2,147,483,646 (円弧補間のみ設定)

b.3 軸連続補間ドライブ用データ構成

```
  [IP_DATA] [CR]
  ;3CIP_No=  終点 1,終点 2,終点 3,速度[CR]
  3CIP_0001=  EndP1 ,  EndP2,  EndP3 , Speed [CR]
  3CIP_0002=  EndP1 ,  EndP2,  EndP3 , Speed [CR]
  ·····  =  ···  ,  ···  ,  ···  ,  ···  [CR]
```

データ	名 称	内 容
[IP_DATA]	セクション	データのヘッダ、先頭に記述
3CIP_nnnn=	3 軸連続補間キー	nnnn : 連続補間ドライブのセグメント番号(0～9999)
EndP1	終点(第 1 軸)	-2,147,483,646～+2,147,483,646
EndP2	終点(第 2 軸)	-2,147,483,646～+2,147,483,646
EndP3	終点(第 3 軸)	-2,147,483,646～+2,147,483,646
Speed	速度パラメータ	0～8000、0:前セグメントと同じ速度 (詳細は 2 軸連続補間と同様)

c.2 軸ビットパターン補間ドライブ

```
[IP_DATA] [CR]
  ;2BP_No=   BP1P,  BP1M,  BP2P,  BP2M[CR]
  2BP_0001= Bp1p ,  Bp1m ,  Bp2p ,  Bp2m [CR]
  2BP_0002= Bp1p ,  Bp1m ,  Bp2p ,  Bp2m [CR]
    ·····  ···  ,  ··· ,  ···  ,  ··· [CR]
```

データ	名 称	内 容
[IP_DATA]	セクション	データのヘッダ、先頭に記述
2BP_nnnn=	2 軸 BP 補間キー	nnnn : BP 補間ドライブのセグメント番号(0～9999)
Bp1p	第 1 軸＋方向 BP	符合なし 16 ビット整数、16 進
Bp1m	第 1 軸－方向 BP	符合なし 16 ビット整数、16 進
Bp2p	第 2 軸＋方向 BP	符合なし 16 ビット整数、16 進
Bp2m	第 2 軸－方向 BP	符合なし 16 ビット整数、16 進

d.3 軸ビットパターン補間ドライブ

```
[IP_DATA] [CR]
;3BP_No=   BP1P,  BP1M,  BP2P,  BP2M,  BP3P,  BP3M [CR]
3BP_0001= Bp1p ,  Bp1m ,  Bp2p ,  Bp2m ,  Bp3p ,  Bp3m [CR]
3BP_0002= Bp1p ,  Bp1m ,  Bp2p ,  Bp2m ,  Bp3p ,  Bp3m [CR]
  ......  ... ,   ... ,  ... ,   ... ,  ... ,   ... [CR]
```

データ	名 称	内 容
[IP_DATA]	セクション	データのヘッダ、先頭に記述
3BP_nnnn=	2 軸 BP 補間キー	nnnn : BP 補間ドライブのセグメント番号(0～9999)
Bp1p	第 1 軸＋方向 BP	符合なし 16 ビット整数、16 進
Bp1m	第 1 軸－方向 BP	符合なし 16 ビット整数、16 進
Bp2p	第 2 軸＋方向 BP	符合なし 16 ビット整数、16 進
Bp2m	第 2 軸－方向 BP	符合なし 16 ビット整数、16 進
Bp3p	第 3 軸＋方向 BP	符合なし 16 ビット整数、16 進
Bp3m	第 3 軸－方向 BP	符合なし 16 ビット整数、16 進

（３）[処理２]連続補間データの書き込み

連続補間データファイルを作成した後は、本製品のフラッシュメモリ(4Mbit)への書き込みを行います。
複数の連続補間データが登録が可能で、連続補間データ番号を付与して書き込みを行います。
連続補間データ番号の設定範囲は 1～65535 で、保存可能な連続補間データ数は、補間セグメント数に換算して合計約 25,000 件です。データの書き込みと削除で使用される API 関数は、次の通りです。

API	内 容
Nmc_2BPWriteEEPROM	2 軸 BP 補間データを書き込む
Nmc_3BPWriteEEPROM	3 軸 BP 補間データを書き込む
Nmc_2CipWriteEEPROM	2 軸連続補間データを書き込む
Nmc_3CipWriteEEPROM	3 軸連続補間データを書き込む
Nmc_2BPReadEEPROM	2 軸 BP 補間データを読み出す
Nmc_3BPReadEEPROM	3 軸 BP 補間データを読み出す
Nmc_2CipReadEEPROM	2 軸連続補間データを読み出す
Nmc_3CipReadEEPROM	3 軸連続補間データを読み出す
Nmc_DeleteEEPROM	連続補間データを削除する

（４）[処理３]連続補間ドライブのモード設定

モード設定では、補間軸の指定、初速度、ドライブ速度、連続補間割り込み／ビットパターン補間割り込みを設定します。
ドライブ速度設定は、先頭の補間セグメントの速度設定です。

1つの連続補間ドライブにおいて、2軸連続補間ドライブは、2軸直線補間ドライブと円弧補間ドライブの混在が可能ですが、他の連続補間ドライブでは、同種のみの補間ドライブの連続となります。

API	内 容
Nmc_WriteReg5( )	補間軸の指定(3,2,1 軸) 2 軸／3 軸直線補間と円弧補間の連続補間ドライブでは、CIINT 割り込みを許可とする 2 軸／3 軸ビットパターン補間ドライブでは、BPINT 割り込みを許可とする 2 軸線速一定／3 軸線速一定の指定は選択とする
Nmc_StartSpd( )	初速度設定
Nmc_Speed( )	ドライブ速度設定

補間関数で指定する補間軸(IpAxis)の指定方法は下記の通りです。
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。ここで指定する補間軸と前節の連続補間データで指定する補間軸は、同一の軸を設定して下さい。
異なる軸を設定すると不定な動作となりますので、ご注意ください。

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
BPINT	CIINT					SPD1	SPD0			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	軸コード(2 進数)
X	00
Y	01
Z	10
U	11

[設定例] 第 3 軸:Z、第 2 軸:Y、第 1 軸:X 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。
D9,8　SPD1,SPD0　00:線速一定無効、 01:2 軸線速一定、 11:3 軸線速一定
D14　CIINT　2 軸連続補間ドライブと 3 軸連続補間ドライブにおいては、'1'をセットします。
D15　BPINT　2 軸ビットパターン補間と 3 軸ビットパターン補間においては、'1'をセットします。

（５）[処理４]連続補間ドライブの実行

連続補間ドライブの実行は、"連続補間データ番号"で指定し、補間実行命令で連続補間ドライブを開始します。
補間実行／停止／状態取得で使用する API 関数は、次の通りです。

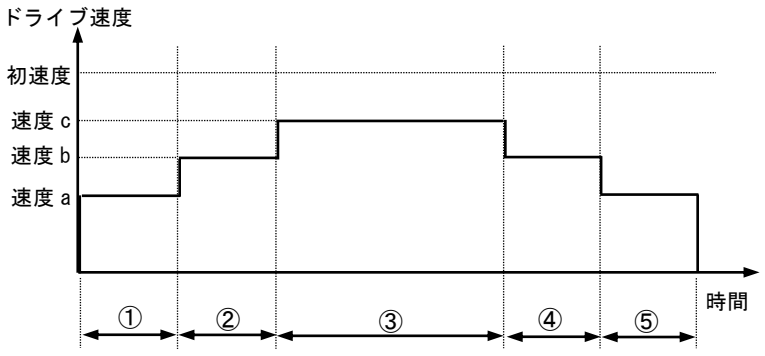
API	内 容
Nmc_Exec_Ip	補間を実行する
Nmc_Stop_Ip	補間を停止する
Nmc_StsRead_Ip	補間実行状態を取得する

（６）速度設定と制限

連続補間ドライブでは、各補間セグメントごとに速度変更を行う事ができますが、自動減速ができません。また、マニュアル減速点の設定もできませんので、加減速ドライブを行う場合は、減速時に速度の調整が必要です。
補間ドライブごとの速度は、連続補間データで指定します。前補間セグメントと同じ場合は速度値'0'を設定します。実際の速度変更は、該当の補間セグメントが開始した直後に設定されて指定の速度となります。

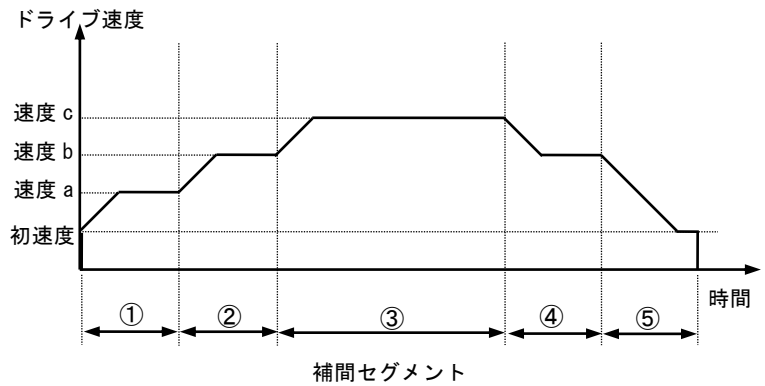
a. 定速ドライブで連続補間ドライブを行う場合（初速度=8000を設定）

速度変更を行うと、ドライブ速度は即指定の速度に変化し、ドライブ終了時は即停止となります。



b. 直線加減速ドライブで連続補間ドライブを行う場合

補間セグメントが変わるとすぐに、指定の速度・加速度・減速度に従って加減速を開始します。
ドライブ終了時には、自動減速ができませんので、最終補間セグメントでは、初速度を設定、出力ドライブパルス数が、初速度に到達できるパルス数に調整する事をお奨めします。



4.7 アプリケーションから制御する補間ドライブ

アプリケーションから補間ドライブ命令(本取扱説明書の3.4節(6)を参照)を指定して実行する事により、単独(1つ)の補間ドライブができます。また、複数の補間ドライブを停止せずに連続して実行する連続補間ドライブは、1つの補間ドライブの実行時間が長い場合(動作環境により異なりますが、約1sec程度)に可能です。
ビットパターン補間ドライブにおいては、48ドライブパルスの出力中に、その後に出力するビットパターンデータを補充する必要がありますので、遅いドライブ速度での実行となります。
ビットパターン補間ドライブの処理は、複雑になりますので、MCX314As/AL 取扱説明の 2.4.3 節を参照して下さい。

(1)機能概要

アプリケーションから制御する単独補間ドライブを実行は、次の手順で行います。
[処理1]補間ドライブのモード設定: 補間軸の指定、補間速度設定、割り込み設定等を行います。
[処理2]補間ドライブ実行　　: 実行命令を発行すると、補間ドライブが実行されます。

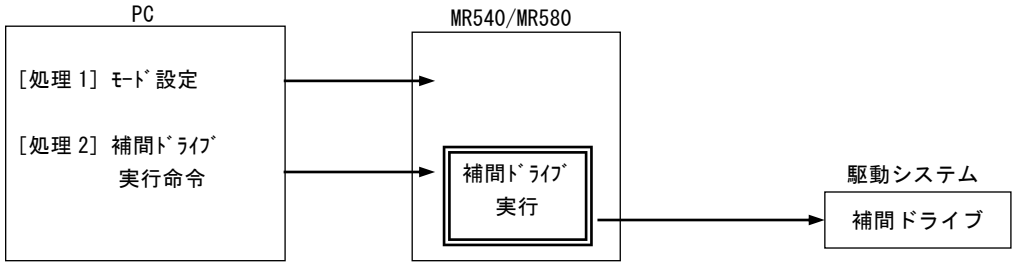


図 4.7-1　単独補間ドライブの操作方法

複数の補間ドライブを停止せずに連続して実行する連続補間ドライブ処理は、現在の補間ドライブ動作中に次の補間ドライブの設定を行います。次の補間ドライブの設定が間に合わない場合は、補間ドライブが一瞬停止します。
正常な連続補間ドライブには、現在の補間ドライブの実行時間や、Windows やデバイスドライバ、アプリケーションなどの処理時間が影響しますので、ご注意ください。1 つの補間ドライブの実行時間が、数 msec 程度の微小な補間ドライブの場合は、ファームウェアの制御する連続補間ドライブ(本取扱説明書の 4.6 節)を参照して下さい。
連続補間ドライブの実行は、次の手順で行います。
[処理1]補間ドライブのモード設定: 補間軸の指定、補間速度設定、割り込み設定等を行います。
[処理2]補間ドライブ実行　　: ①用実行命令を発行すると、補間ドライブ①が実行されます。
[処理3]次の補間ドライブ設定　: 次の補間ドライブデータ書き込みが可能か確認し、可能であれば書き込みます。

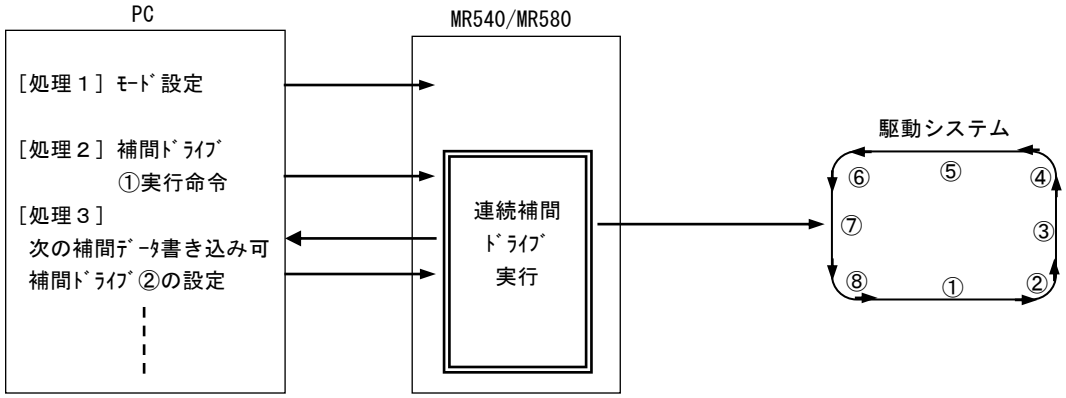


図 4.7-2　連続補間ドライブの操作方法

補間ドライブの制限事項と注意事項は、次の通りです。
・MR580 の場合は、メイン基板と増設 4 軸基板のそれぞれの基板内の軸間で補間ドライブを行います。
　異なるユニットの軸間では、補間ドライブができませんので、ご注意ください。
・MR580 の場合は、メイン基板と増設 4 軸基板において、同時に 2 個の連続補間ドライブはできます。
・連続補間ドライブの場合は、現在の補間ドライブ動作中に次の補間ドライブの設定が必要ですので、ご注意ください。
・連続補間ドライブの停止は、Nmc_Stop_Ip()関数を使用して下さい。
・MR540 を使用する際、関数の IC 番号の引数には、'0'を必ず指定して下さい。
詳細は、MCX314As/AL取扱説明書の2.4節を参照して下さい。

（２）[処理１]補間ドライブのモード設定

モード設定では、補間軸の指定、初速度、ドライブ速度、連続補間割り込み／ビットパターン補間割り込みを設定します。
直線補間ドライブでは、終点の設定を行い、円弧補間ドライブでは、終点と中心点の設定を行います。
連続補間ドライブにおいては、次の補間ドライブの速度は設定できません。速度を設定した時点で変更となりますので、ご注意ください。

API	内 容
Nmc_WriteReg5( )	補間軸の指定(3,2,1 軸) 2 軸／3 軸直線補間と円弧補間の連続補間ドライブの際の CIINT 割り込み・許可／不許可 2 軸線速一定／3 軸線速一定の指定は選択とする
Nmc_StartSpd( )	初速度設定
Nmc_Speed( )	ドライブ速度設定 （連続補間ドライブの場合は、設定した時点で速度変更となる）
Nmc_Pulse( )	終点の設定
Nmc_Center( )	円弧補間の中心点

補間軸の指定や割り込みの設定方法は、本取扱説明書の 4.6 節(4)と同様です。
連続補間ドライブにおいて、Nmc_WriteReg5( )のCINT割り込み・許可とした場合、現在の補間ドライブが動作中に、次の補間ドライブデータの書き込みが可能となると割り込みが発生します。

（３）[処理２]補間ドライブの実行

補間ドライブの実行は、次の API 関数で補間ドライブ命令(本取扱説明書 3.4 節(6)を参照)を指定し実行します。

API	内 容
Nmc_Command_IP	補間ドライブ命令を実行する 【主な補間ドライブ命令】 ・#define CMD_IP_2ST 0x30 2 軸直線補間ﾄﾞﾗｲﾌﾞ ・#define CMD_IP_3ST 0x31 3 軸直線補間ﾄﾞﾗｲﾌﾞ ・#define CMD_IP_CW 0x32 CW 円弧補間ﾄﾞﾗｲﾌﾞ ・#define CMD_IP_CCW 0x33 CCW 円弧補間ﾄﾞﾗｲﾌﾞ

（４）[処理３]次の補間データ書き込み（連続補間ドライブの場合）

連続補間ドライブの処理においては、現在、動作している補間ドライブ中に次の補間ドライブデータを書き込みます。
その際、Nmc_GetCNextStatus( )関数で次の補間ドライブデータ書き込みが可能か確認し、可能であれば書き込みます。
書き込み状態取得と次補間ドライブデータを設定する API 関数は、次の通りです。

API	内 容
Nmc_GetCNextStatus	補間次データ書き込み状態取得
Nmc_Command_IP	次補間ドライブ命令を設定
Nmc_Pulse( )	終点の設定
Nmc_Center( )	円弧補間の中心点

Nmc_WriteReg5( )の CINT 割り込み・許可とした場合、次の補間ドライブデータ書き込みが可能となると、割り込みが発生します。その割り込み処理の中で、次の補間ドライブデータを書き込みます。

4.8 同期動作

本機能は MCX314AL の機能で、指定の起動要因が発生したら、MCX314AL が CPU の処理を介在することなく、指定の動作を直ちに行う機能です。従って、精度の高い同期動作が可能となります。PC との通信処理やファームウェアの制御処理により動作タイミングに遅れが発生する場合に、本機能が有効です。
MR580 の場合は、メイン基板の 4 軸間、増設 4 軸基板の 4 軸間で同期動作を行います。異なる基板の軸間では、同期動作ができませんので、ご注意ください。

（1） 機能概要

起動要因と動作の概要は、次の通りです。

	起動要因の発生する軸（自軸）への設定	
起動要因	指定の位置を通過した場合	1. 論理／実位置カウンタが COMP+値を越えて大きくなった
		2. 論理／実位置カウンタが COMP+値を越えて小さくなった
		3. 論理／実位置カウンタが COMP−値を越えて小さくなった
		4. 論理／実位置カウンタが COMP−値を越えて大きくなった
	ドライブの状態が変化した場合	5. 自軸がドライブを開始した
		6. 自軸がドライブを終了した
	nIN3 入力信号が変化した場合 （nECZ 端子を汎用入力として使用）	7. 自軸のnIN3 信号が Low→Hi に立ち上がった
		8. 自軸のnIN3 信号が Hi→Low に立ち下がった
	その他	9. 論理位置カウンタを読み出した （命令(10h)が書き込まれた）
		10. 同期動作の起動を行った （命令(65h)が書き込まれた）

⇩

	動作させる軸（他軸）への設定 （自軸に対しても動作させる場合は自軸にも設定）	
動 作	ドライブ動作	1. ＋方向の定量パルスドライブを起動する
		2. －方向の定量パルスドライブを起動する
		3. ＋方向の連続パルスドライブを起動する
		4. －方向の連続パルスドライブを起動する
		5. ドライブを減速停止させる
		6. ドライブを即停止させる
	MCX314AL のレジスタ処理	7. 現在の論理位置カウンタ値を同期ﾊﾞｯﾌｧにセーブする
		8. 現在の実位置カウンタ値を同期バッファにセーブする
		9. WR6,WR7 ﾚｼﾞｽﾀの値を、論理位置カウンタにｾｯﾄする
		10. WR6,WR7 ﾚｼﾞｽﾀの値を、実位置カウンタにｾｯﾄする
		11. WR6,WR7 ﾚｼﾞｽﾀの値を、出力パルス数にｾｯﾄする
		12. WR6 ﾚｼﾞｽﾀの値を、ドライブ速度にｾｯﾄする
	同期パルスの出力	13. 同期パルスとして nDCC 信号を出力する
	割り込み処理	14. 割り込み信号を発生させる

（2） 設定と実行

本機能の設定は、同期動作モード設定 Nmc_SyncMode( )で行います。起動要因の発生する軸（自軸）に対しての設定と、動作させる軸（他軸）に対しての設定が必要です。動作させる軸が自軸の場合は、他軸の設定は必要ありません。
機能についての詳細は、MCX314As/AL 取扱説明書 2.6.1 節をご参照ください。

API	内 容
Nmc_SyncMode( )	・設定：同期動作に関連するすべての軸に対して設定します。 1. 自軸に対する設定 SM6_data：自軸の起動要因の指定と、動作させる軸を指定する。 SM7_data：自軸に対する動作を指定する（自軸の動作がなければ指定しない）。 2. 他軸に対する設定 SM6_data：起動要因は何も指定しない。 SM7_data：動作を指定する。 ・解除：同期動作の後、繰返し動作が不要の場合は、各軸に対して解除を設定します。 SM6_data： 0 SM7_data： 0

（３） 同期動作の実例

同期動作機能を使用した実例を示します。

項　目	起動要因	動　作
ドライブ制御の同期	Y 軸:位置 15,000 を通過	Z 軸:＋方向定量パルスドライブ開始
ドライブ制御の同期	X 軸:位置−320,000 を通過	Y 軸:ドライブ停止、Z 軸:ドライブ停止
位置情報の同時読み出し	X 軸:XIN3 入力信号の立ち下がり	X,Y,Z 軸:位置データを各同期バッファにセーブ 割り込み発生:CPU は同期バッファを読み出す
位置情報の同時読み出し	X 軸:10h 命令が書き込まれた	X,Y,Z 軸:位置データを各同期バッファにセーブ 割り込み発生:CPU は同期バッファを読み出す
定量ドライブの連続動作	X 軸:ドライブ終了	X 軸:WR6,7－＞出力パルス数(P) ＆ ドライブ起動
ドライブ速度変更	Z 軸:位置 10,000 を通過	Z 軸:WR6－＞ドライブ速度(V)

4.9 入力信号フィルタ

本機能は MCX314AL の機能で、各入力信号ごとの積分型フィルタが設定可能で、信号ごとにノイズ対策ができます。

（１） 機能概要

ノイズ対策などで入力信号の遅延が必要な場合、該当の入力信号にフィルタ有効設定を行い、8種類の時定数からノイズ環境に合った値を選択できます。アプリケーションの最初にコールするNmc_Open()関数では、初期設定として時定数'2'(遅延時間512μsec)に設定しています。入力信号回路の遅延時間は100μsec程度ですので、入力信号フィルタの初期値と合わせると、初期時の入力信号遅延時間は、610μsec程度となります。

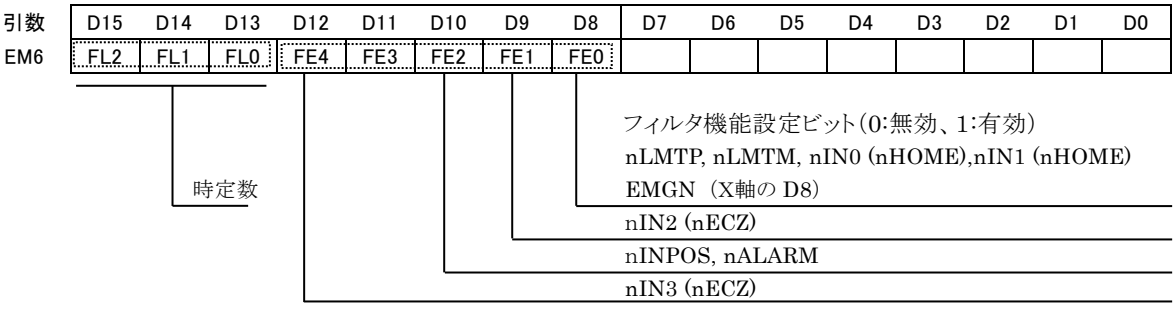
（２） 設　定

フィルタの時定数は、下表に示す8段階の値から選択します。時定数を上げると除去可能な最大ノイズ幅は上がりますが、信号の遅延時間が大きくなりますので、適切な値を設定して下さい。
詳細は、MCX314As/AL取扱説明書2.8節、6.16節を参照して下さい。

API	内　容
Nmc_ExpMode( )	フィルタの有効／無効を設定 フィルタを有効とする場合は、時定数を設定

時定数	除去可能な最大ノイズ幅		入力信号遅延時間	
0	1.75	μsec	2	μsec
1	224	μsec	256	μsec
2	448	μsec	512	μsec
3	896	μsec	1.024	msec
4	1.792	msec	2.048	msec
5	3.584	msec	4.096	msec
6	7.168	msec	8.192	msec
7	14.336	msec	16.384	msec

Nmc_ExpMode( )の引き数EM6_dataの設定方法は、次を参照して下さい。



5. 評価ツール

（１）機能概要

MR540/MR580 評価ツール（以下、評価ツールと称します）では、本製品と PC を接続すればすぐに動作確認ができます。評価ツールを起動すると、ユニット番号選択画面が表示されますので、ユニット番号、通信 I/F、割り込み使用の有無を設定し、DLL 初期化ボタンを押下後、OK ボタンを押下する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、各種ドライブ命令の実行、現在位置・速度等の表示、割り込み画面表示ができます。別の画面では、詳細な機能の設定（モード設定、拡張モード設定、同期動作モード設定画面にて）ができ、詳細な状態表示（ステータス画面にて）ができます。
また、パラメータ設定とモード設定は、PC に保存ができ後で読み出しもできます。

（２）起動方法

本製品と PC を、USB または LAN で接続し、電源を ON します。
そして、MR540_Tool.exe を実行すると、ユニット番号設定画面が表示されます。

① ユニット番号を選択　：接続した本製品のロータリスイッチの番号に合わせます。
② 通信 I/F を選択　：接続している形式（USB または LAN）を選択します。
③ 割り込み使用の有無　：割り込みを使用する場合は、“割り込み有効”をチェックします。
④ DLL の初期化　：本製品の DLL を初期化します
⑤ OK を押下　：通信が正常に行われるとメイン画面が表示されます。

（３）メイン画面

メイン画面では、次の操作を行います。

① ユニット表示 ： ユニット番号とユニット種別（MR540/MR580）、モーションコントロール IC 名を表示します。

② 現在状態表示 ： 軸ごとの現在の位置カウンタ値や現在のドライブ速度値を表示します。
論理位置カウンタと実位置カウンタは、入力により設定ができ、[CL]押下で値のクリアもできます。メイン基板の状態は IC0 に、増設 4 軸基板の状態は IC1 に表示します。

③ パラメータ設定 ： 軸ごとに各種パラメータを設定します。
メイン基板の設定は IC0 に、増設 4 軸基板の設定は IC1 に表示します。

④ ドライブ軸の指定 ： IC 選択と軸指定によりドライブする軸を指定します。
割り込みを使用する場合は、[ユーザー関数設定]にチェックを入れます。

⑤ ドライブ命令 ： 上記④項で指定された軸に対してドライブ開始や停止を実行します。
指定した軸に対して同じコマンドが発行されます。

⑥ 補間命令 ： アプリケーションから制御する補間ドライブを実行します。補間軸は、モード設定の WR5 設定で行い、終点や円弧中心点は上記③項で行います。

⑦ BP 補間設定 ： ビットパターン補間ドライブのビットパターン（16Bit）を設定します。
初めに[36]押下で、BP レジスタ書き込み可能として、ビットパターンを入力します。
次に[38]押下で、入力されたビットパターンを IC 内部のレジスタに書き込みます。
1 個の IC 内部レジスタに書き込まれると‘1’表示で、最大で 3 個まで書き込むことができます。ビットパターンの内容は、各内部レジスタともに同じです。
最後に[37]押下で、BP レジスタ書き込み不可とした後に、上記⑥項の BP 補間命令を実行します。

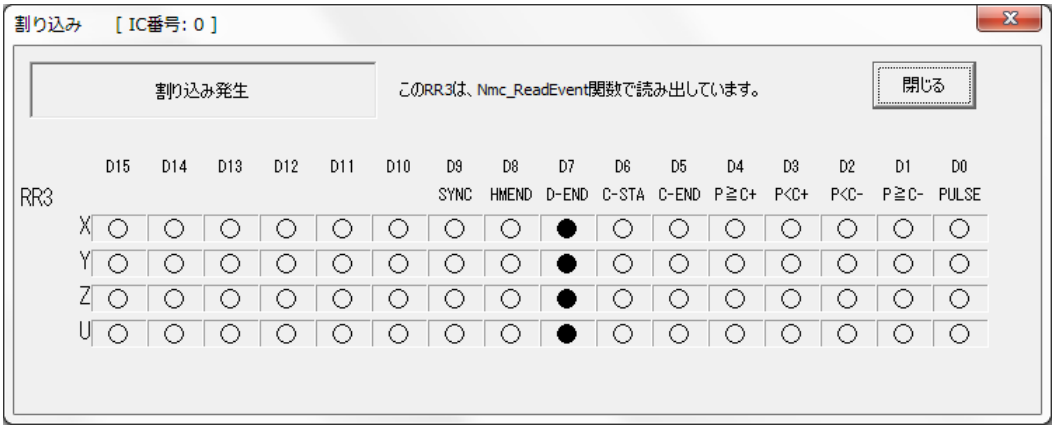
⑧ ステータス表示 ： [ステータス]押下で、MCX314AL リードレジスタの状態が表示されます。

⑨ 各モード設定 ： [モード設定]、[拡張モード設定]、[同期動作モード設定]のボタン押下で、それぞれの設定画面が表示されます。

⑩ ロード/セーブ ： パラメータ設定、モード設定、拡張モード設定、同期動作モード設定、ビットパターンデータをファイルに保存して、その後読み出すことができます。

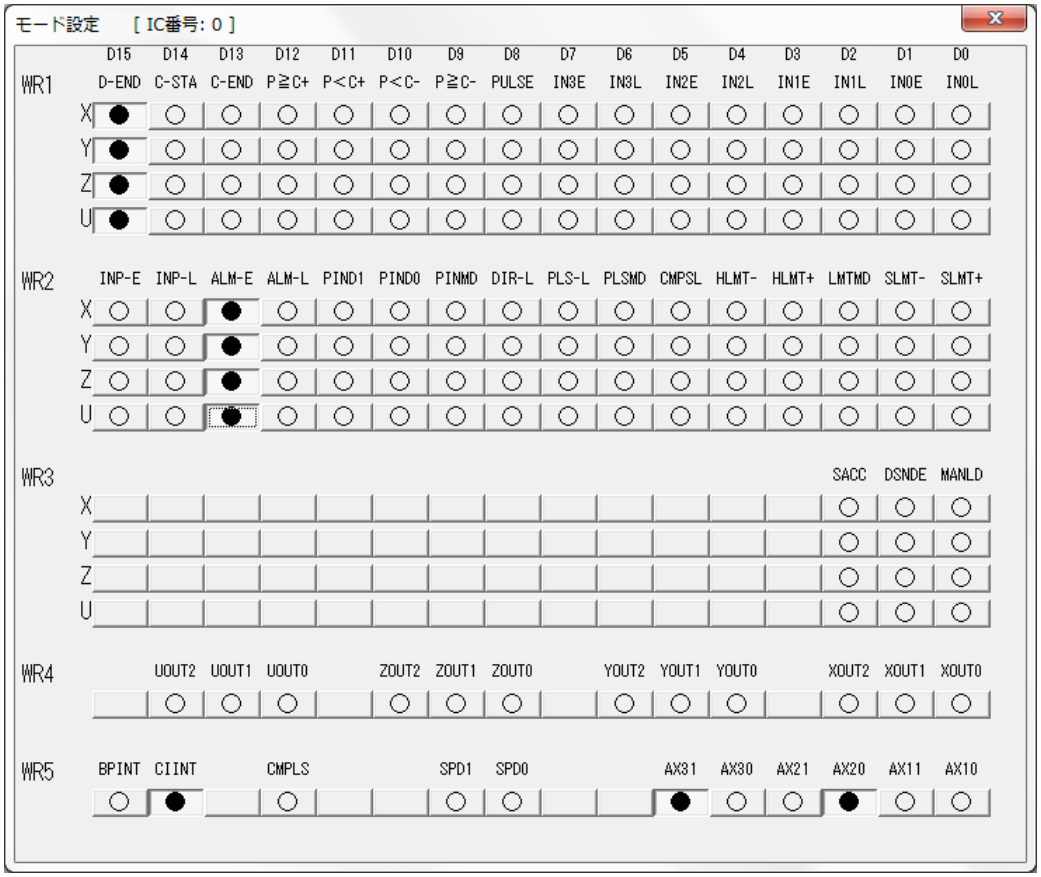
（４）割り込み画面

割り込みを有効に設定して、割り込みが発生すると「割り込み」画面に、割り込み要因が表示されます。
割り込みの有効設定は、起動時の「ユニット番号選択」画面で“割り込み有効”とし、メイン画面の軸指定で「ユーザー関数設定」にチェックを入れて、「モード設定」画面で指定の割り込みを有効に設定します。
割り込みが、同時に複数発生した場合は、1つの画面に複数の割り込み要因が表示され、異なるタイミングで発生した場合は、複数の割り込み要因表示画面が表示されます。



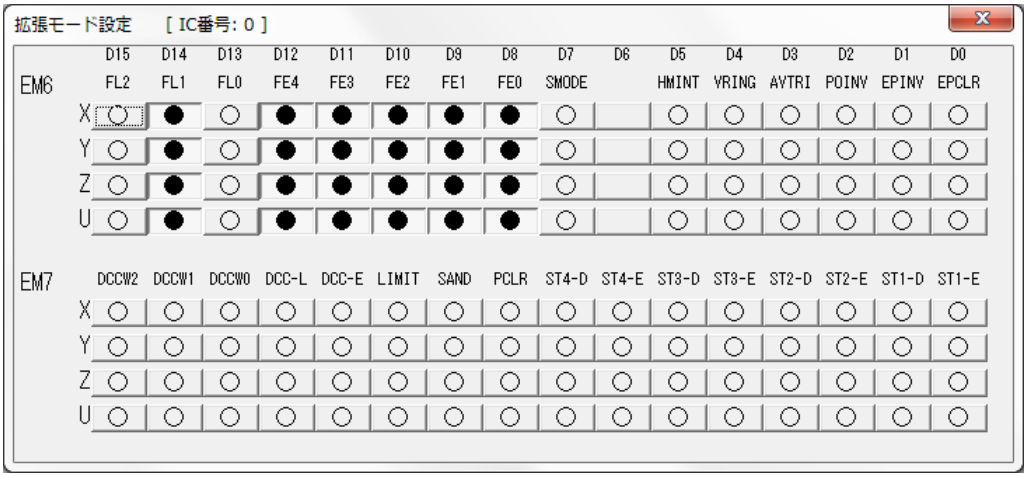
（５）モード設定画面

「モード設定」画面では、MCX314AL の WR1～WR5（モードレジスタ 1,2,3、アウトプットレジスタ、補間モードレジスタ）の設定を行います。
WR1～WR3,WR5（モードレジスタ 1,2,3、補間モードレジスタ）の場合は、○=0、●=1 を示します。
WR4（アウトプットレジスタ）の場合は、○=出力 OFF、●=出力 ON を示します。
各ビットの内容につきましては、MCX314As/AL 取扱説明書を参照して下さい。



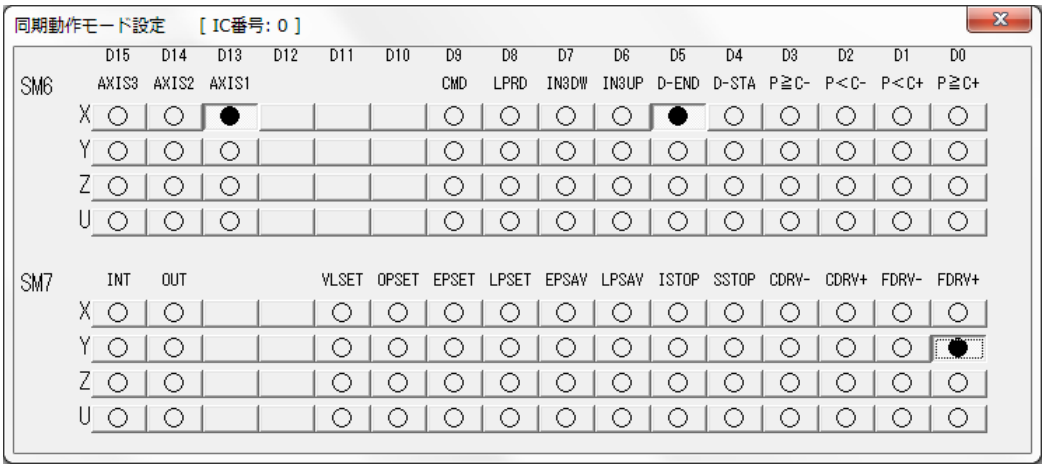
（６）拡張モード設定画面

「拡張モード設定」画面では、MCX314AL の拡張モードレジスタ EM6,EM7 の設定を行います。
○=0、●=1 を示します。



（７）同期動作モード設定画面

「同期動作モード設定」画面では、MCX314AL の同期動作モードレジスタ SM6,SM7 の設定を行います。
○=0、●=1 を示します。

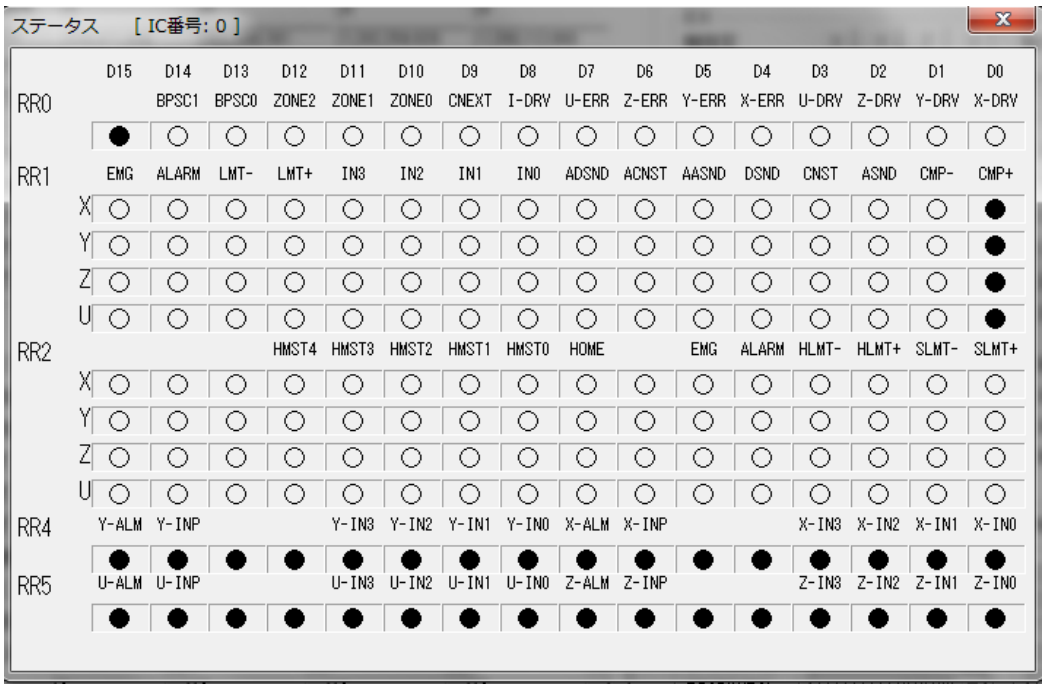


（８）ステータス画面

「ステータス」画面では、MCX314AL の RR0,1,2,4,5（主ステータスレジスタ、ステータスレジスタ 1,2、インプットレジスタ 1,2）の内容を、100msec 間隔で読み出します。RR3（ステータスレジスタ 3）につきましては、割り込み発生時の「割り込み」画面に表示します。

RR0～RR2（主ステータスレジスタ、ステータスレジスタ 1,2）の場合は、○=0、●=1 を示します。

RR4,RR5（インプットレジスタ 1,2）の場合は、○=入力信号端子が GND と短絡し入力 ON 状態、●=入力信号端子がオープンで入力 OFF 状態を示します。



■Windows は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。