ＰＣＩバス/ ＰＣＩＥｘｐｒｅｓｓ対応
モーションコントロールボード

MC8000P
デバイスドライバ
取扱説明書

初版　　２００６．１１．１６　[Ver 1.0.0.0]
改訂　　２０１３．１１．０６　[Ver 6.0.0.0]

対応ボード
MC8043P/ MC8043Pe
MC8082P/ MC8082Pe
MC8022P
MC8042P

NOVA electronics　　株式会社 ノヴァエレクトロニクス

はじめに

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

■安全にお使いいただくために

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

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

■マニュアルの併用

ボードの仕様などについては、ハードウェア取扱説明書を参照してください。

■本書で使用する特殊用語

ＩＣ　　ＩＣとはMCX314As、MCX304、MCX514の事を指します。

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

目　次

1. 概要

本デバイスドライバは、ノヴァエレクトロニクス製ＰＣＩバス/ＰＣＩ　Ｅｘｐｒｅｓｓ対応モータコントロールボード共通デバイスドライバ　　です。
ボードの仕様は、各ボードの取扱説明書と各ボードに搭載しているＩＣの取扱説明書を参照して下さい。

1.1 対応ボード

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

対応ボード
MC8082P / MC8082Pe
MC8042P
MC8022P
MC8080P
MC8043P / MC8043Pe

1.2 対応ＯＳ

本デバイスドライバは下記のＯＳに対応しています。

対応ＯＳ
Windows98(32bit)
Windows2000(32bit)
WindowsXP(32bit)
WindowsVista(32bit,64bit)
Windows7(32bit,64bit)

1.3 対応言語

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

対応言語
Microsoft Visual C++ 6.0
Microsoft Visual C++ .NET 2003
Microsoft Visual C++ 2005
Microsoft Visual C++ 2008
Microsoft Visual Basic 6.0
Microsoft Visual Basic .NET 2003
Microsoft Visual Basic 2005
Microsoft Visual Basic 2008
Microsoft Visual C# .NET 2003
Microsoft Visual C# 2005
Microsoft Visual C# 2008

注）言語に対応する開発環境とＯＳとの関係はMicrosoftのホームページを参照ください。

1.4 最大ボード数

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

1.5 ボード搭載ＩＣ

対応ボードに搭載しているＩＣは下記の通りです。

ボード	搭載ＩＣ	ＩＣ数	軸数
MC8082P / MC8082Pe	MCX304	2	8
MC8042P	MCX304	1	4
MC8022P	MCX304	1	2
MC8080P	MCX304	2	8
MC8043P / MC8043Pe	MCX314As	1	4

２．インストール

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

2.1 ドライバソフトウェアの準備

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

2.2 本ボードを複数枚使用する場合の設定

本デバイスドライバは本ボードを同時に１６枚まで認識します。

本ボードを１つのシステム（ＰＣ）で複数枚使用する時は、ボードを個別に認識させる為に、２枚目以降のボードはボード番号を
ボード上のロータリスイッチで設定して下さい。
ロータリスイッチ（ＳＷ１）の位置は、各ボードの取扱説明書「基板外形」の章を参照してください。
ロータリスイッチは０～Ｆのいずれかを設定できます。ロータリースイッチの番号は他のボードと重複しないように設定して下さい。

出荷時は、ボード上のロータリスイッチに０が設定されています。

2.3 パソコンへの本ボードの組込み

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

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

2.4 デバイスドライバのインストール

各ＯＳに対応したインストール手順を説明します。

2.4.1 Windows 98 (32bit)

① 2.1の方法でインストールするデバイスドライバを準備して下さい。

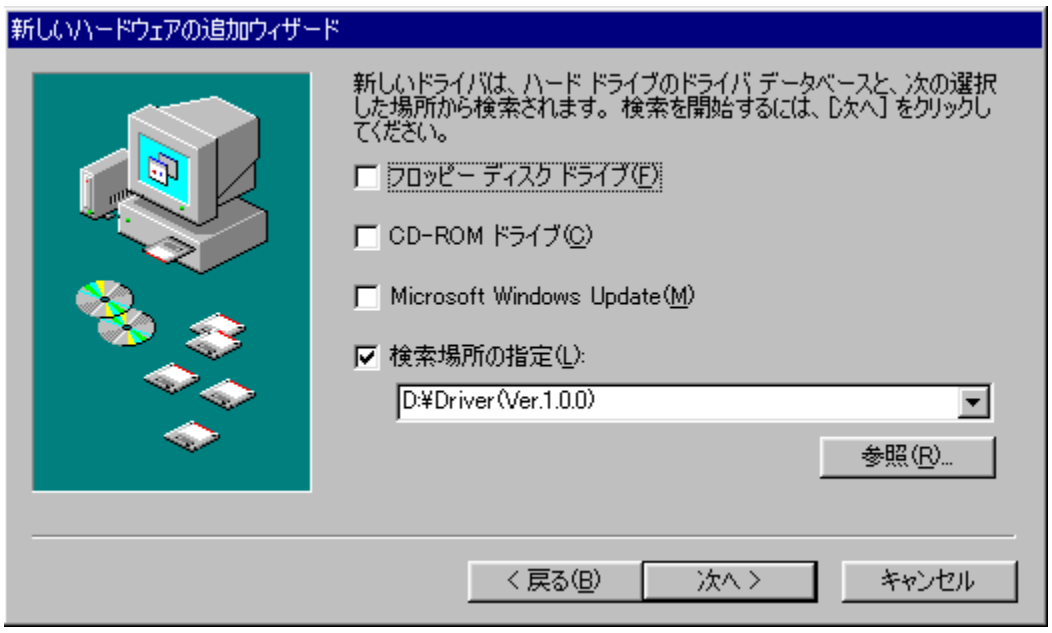
② 2.2, 2.3 を実行し、本ボードが確実にパソコンに組み込まれているか確認してください。

③ パソコン本体の電源をＯＮし、Windows 98を起動します。

④ すぐに「新しいハードウェア」と表示され「ハードウェア追加ウィザード」が起動しますので、[次へ]をクリックします。

⑤ デバイスドライバの検出方法を選択する画面が表示されますので、「使用中のデバイスに最適なドライバを検索する(推奨)」にチェックをして[次へ]をクリックします。

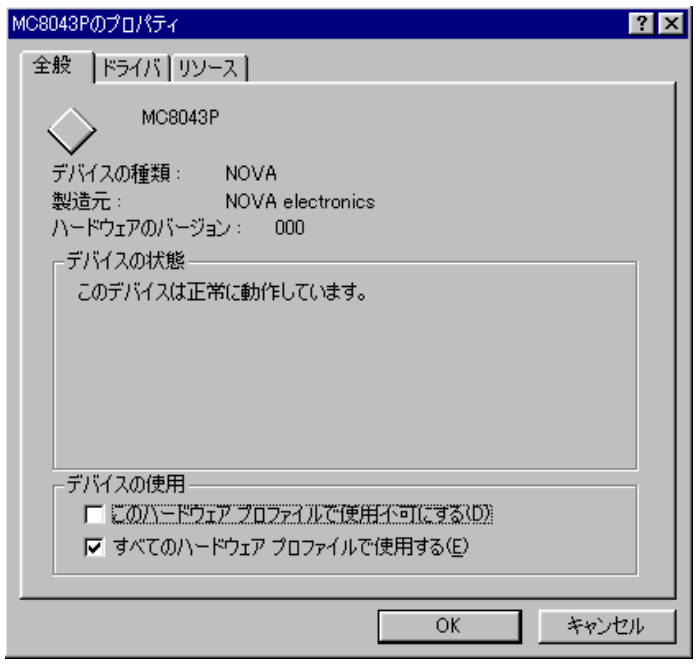
⑥　検索場所の指定画面が表示されますので[検索場所の指定]をチェックします。
ドライバをＣＤ－ＲＯＭからインストールする場合は、パソコンにＣＤ－ＲＯＭをセットし、ＣＤ－ＲＯＭが認識されてから次の動作に進みます。
参照ボタンを押し、提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、テキストボックスにフォルダが表示されたら、[次へ]をクリックします。



⑦　ボード名（MC8043Pなど）が表示されたら[次へ]をクリックします。

⑧　デバイスドライバのコピーが完了すると､メッセージが表示されますので[完了]をクリックします。

⑨　以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システム」－「デバイスマネージャ」画面（下記左画面）を開き、
「ＮＯＶＡ」の下のボード名（ＭＣ８０４３Ｐなど）をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。



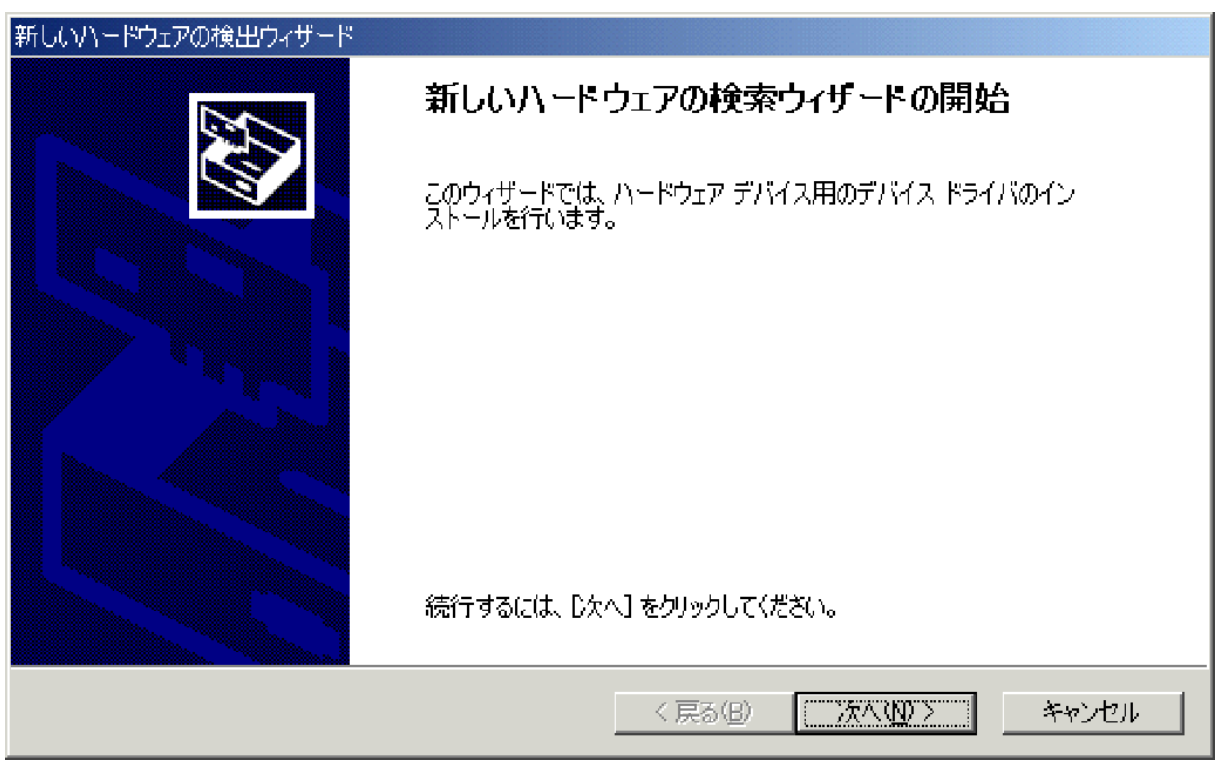
また、デバイスドライバのインストール完了後はパソコンの起動時に手順④のようにハードウェアウィザードが起動することはありません｡もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので､その場合は2.5の手順に従って一度本ボードを取り外した後に2.3の手順から再度インストールをやり直してください。

インストールを完了したらリソース(Ｉ／Ｏアドレス、割り込みレベル)の設定､競合の有無を[コントロールパネル]－[システム]－[デバイスマネージャ]でボード名をダブルクリックし、[リソース]タブで確認してくだい。

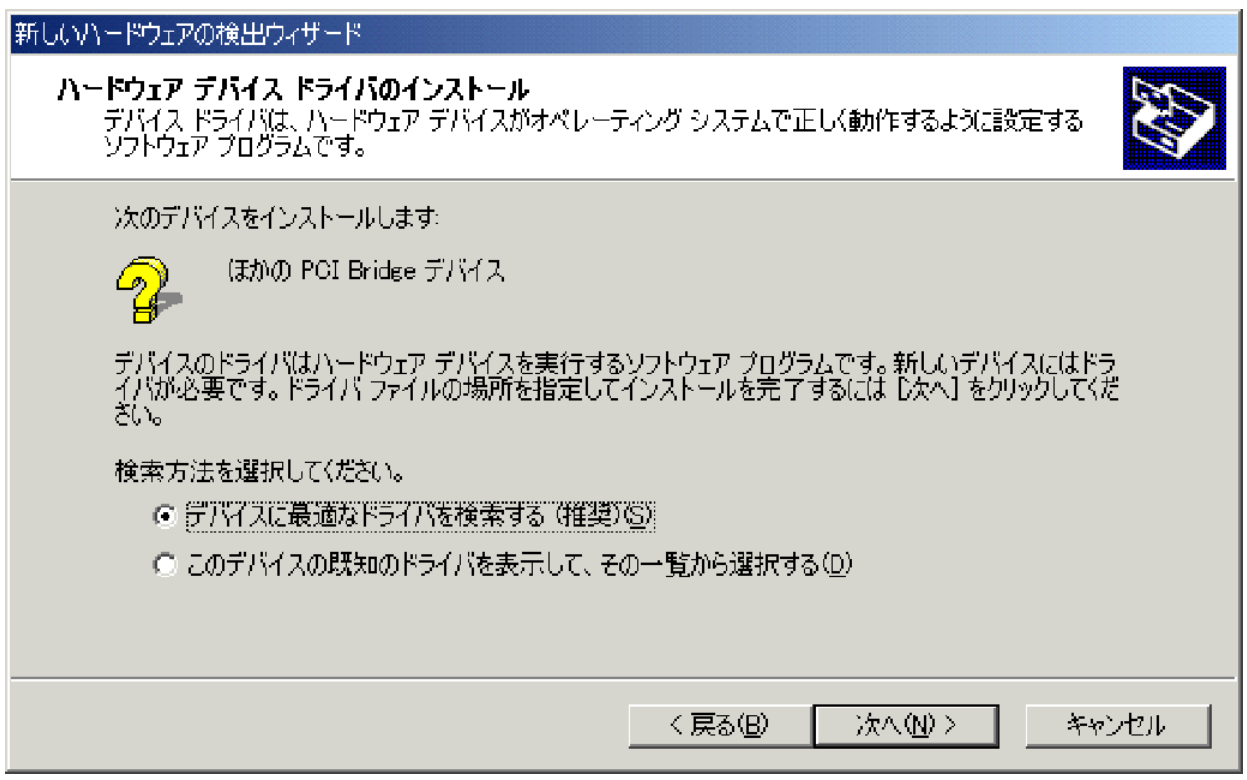
2.4.2Windows 2000 (32bit)

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

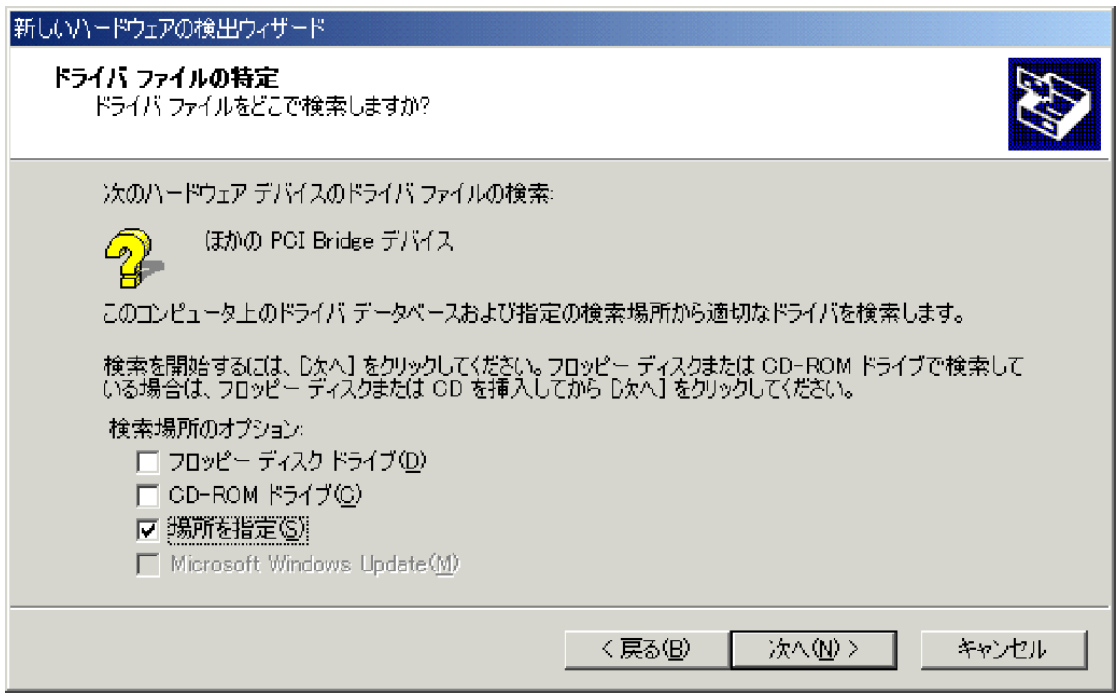
① 2.1の方法でインストールするデバイスドライバを準備して下さい。
② 2.2, 2.3 を実行し、本ボードが確実にパソコンに組み込まれているか確認してください。
③ パソコン本体の電源をＯＮし、Windows 2000を起動します。
④ アドミニストレーター権限を持ったユーザーでログインしてください。
⑤ すぐに「新しいハードウェアが見つかりました」と表示されデバイスドライバのインストールが開始されます。
⑥ すぐに「新しいハードウェアの検出ウィザード」が表示され「新しいハードウェアの検索ウィザードの開始」が出ますので[次へ]をクリックします。



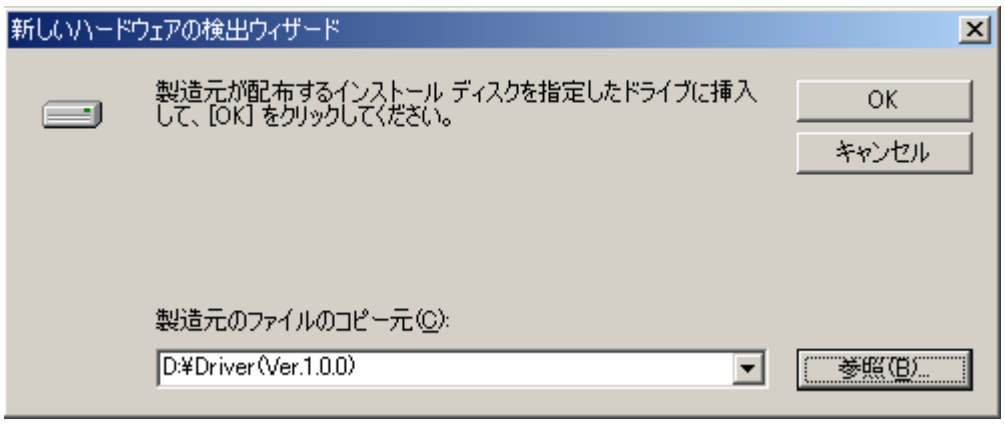
⑦ 「ハードウェアデバイスドライバのインストール」が表示され、デバイスドライバの自動検出を促す画面が出ますので、「デバイスに最適なドライバを検出する(推奨)」にチェックして[次へ]をクリックします。



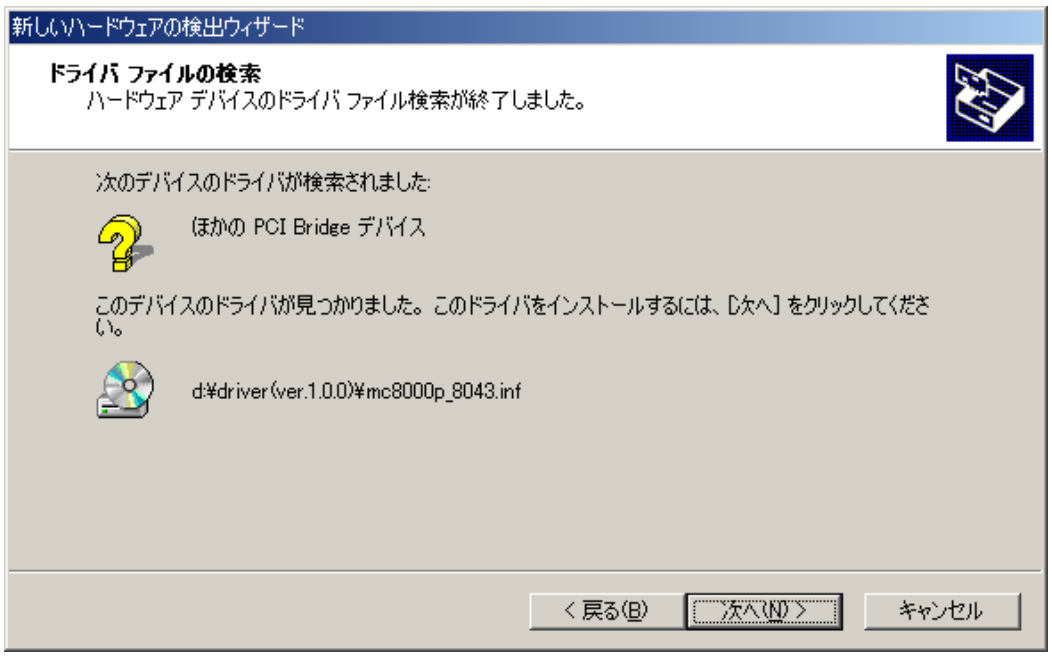
⑧　検索場所の指定が出ますので[場所を指定]にチェックして[次へ]をクリックします。



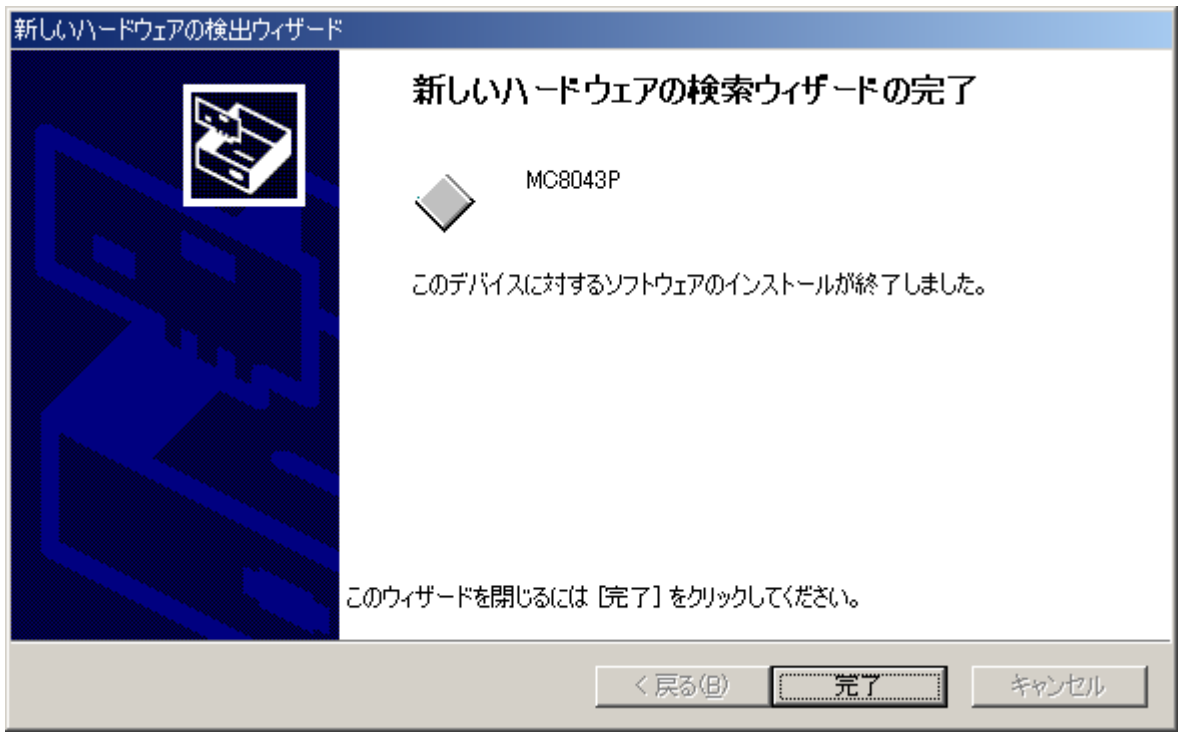
⑨　下のような画面が出ます。
ドライバをＣＤ－ＲＯＭからインストールする場合は、パソコンにＣＤ－ＲＯＭをセットし、ＣＤ－ＲＯＭが認識されてから次の動作に進みます。
参照ボタンを押し、提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、テキストボックスにフォルダが表示されたら、
ＯＫを押してください。



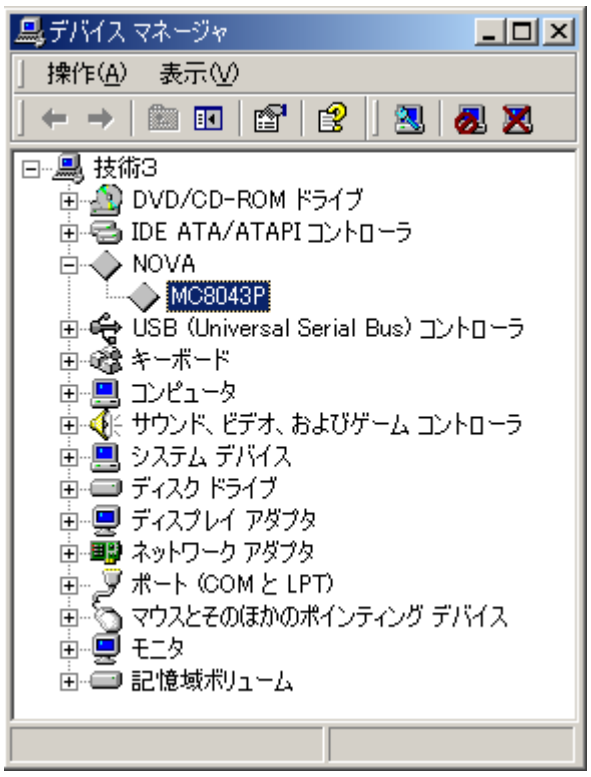
⑩　デバイスドライバの情報ファイルが発見されると確認の画面が出ますので、[次へ]をクリックします。

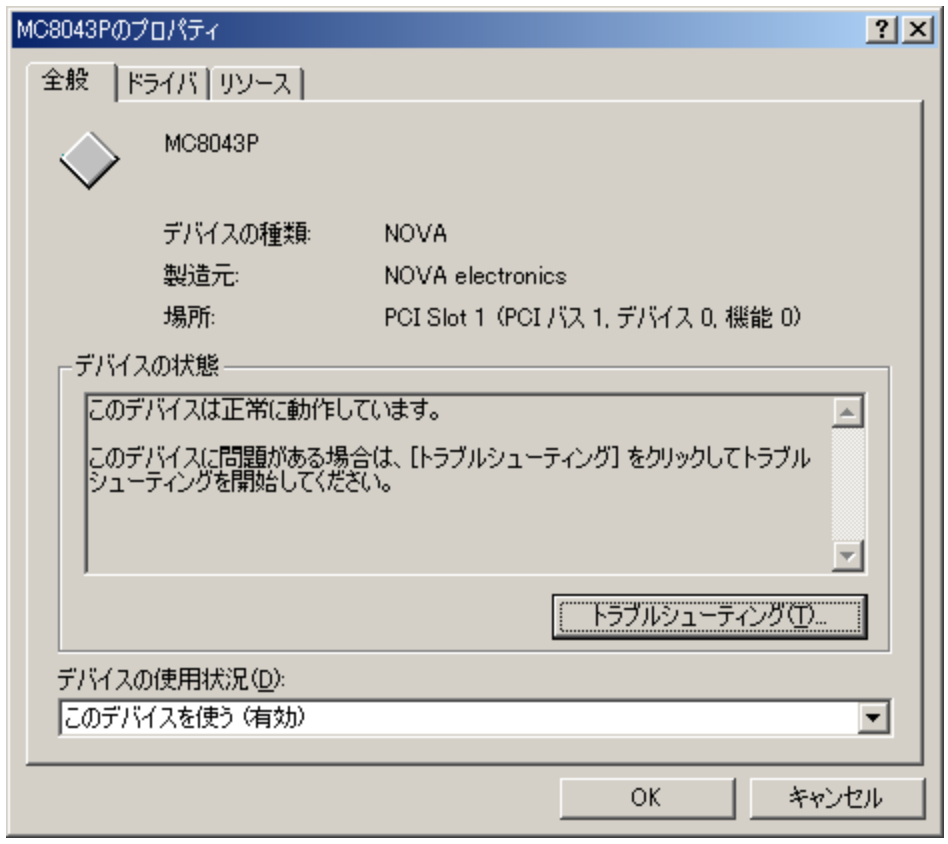


⑪　デバイスドライバのインストールが正常に完了すると、メッセージが出ますので、[完了]をクリックします。



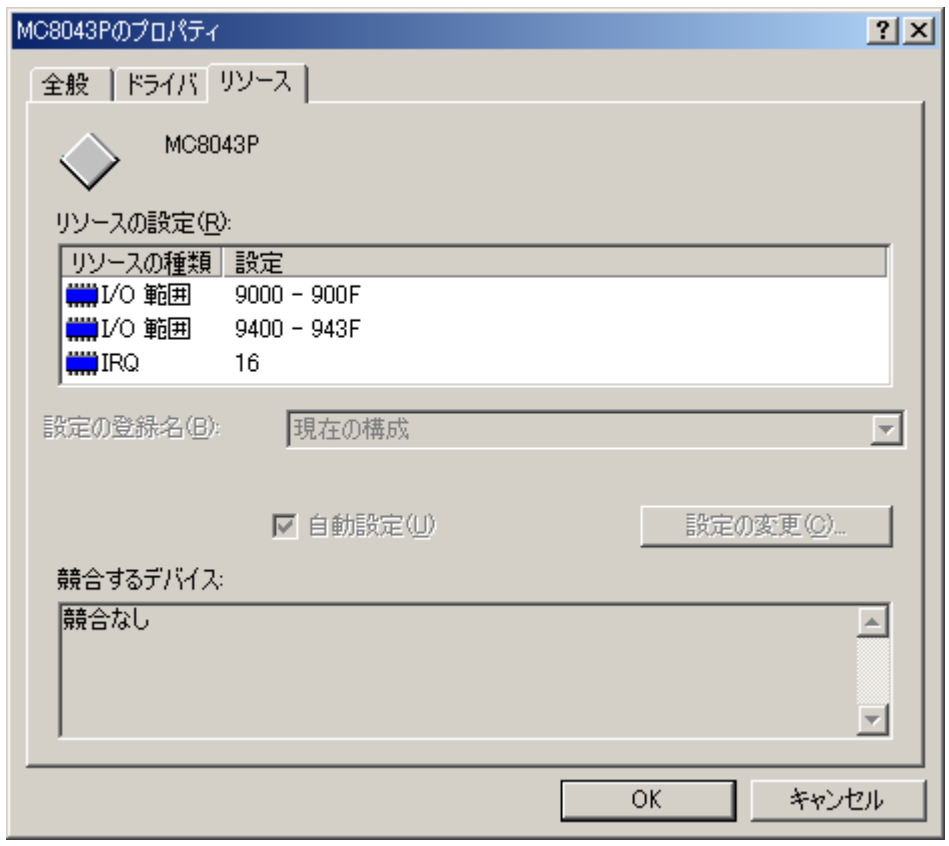
⑫　以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、
「ＮＯＶＡ」の下のボード名（ＭＣ８０４３Ｐなど）をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。





また、デバイスドライバのインストール完了後はパソコンの起動時に手順⑥のようにハードウェアウィザードが起動することはありません。もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので、その場合は2.5の手順に従って一度本ボードを取り外した後に2.3の手順から再度インストールをやり直してください。

インストールを完了したらリソース(Ｉ／Ｏアドレス、割り込みレベル)の設定､競合の有無を[コントロールパネル]－[システム]－[ハードウェア]－[デバイスマネージャ]でボード名をダブルクリックし、[リソース]タブで確認してください。

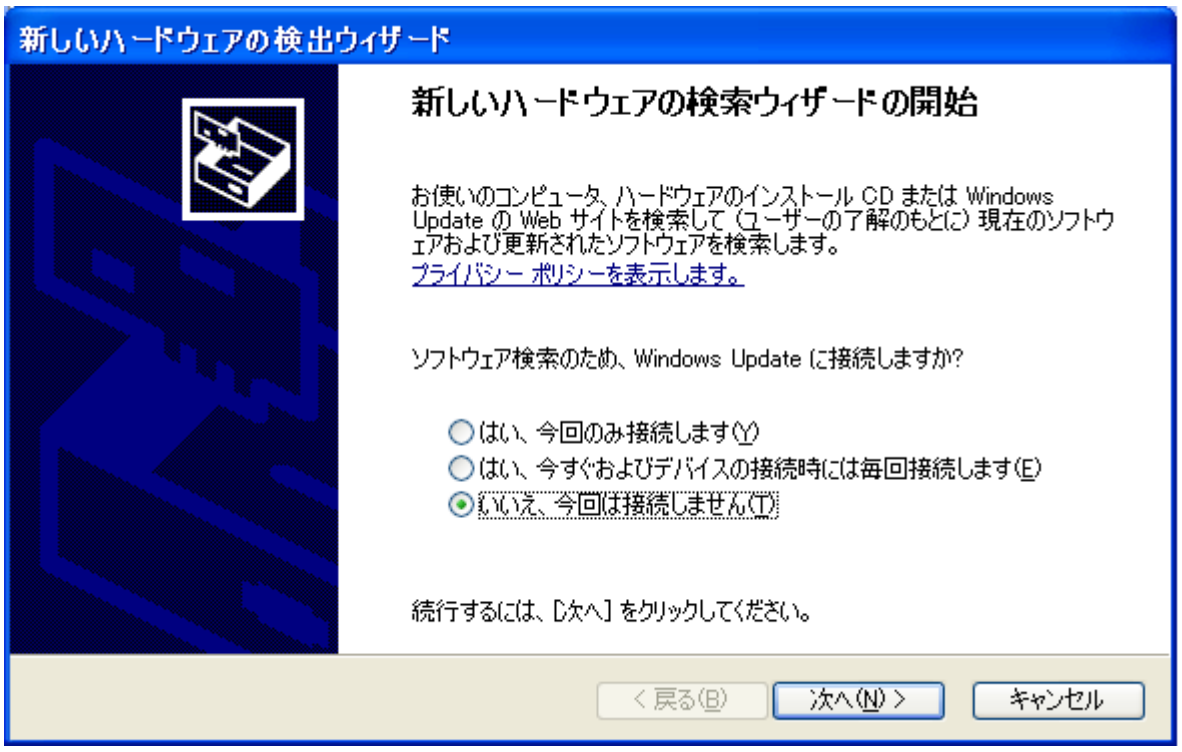


2.4.3Windows XP (32bit)

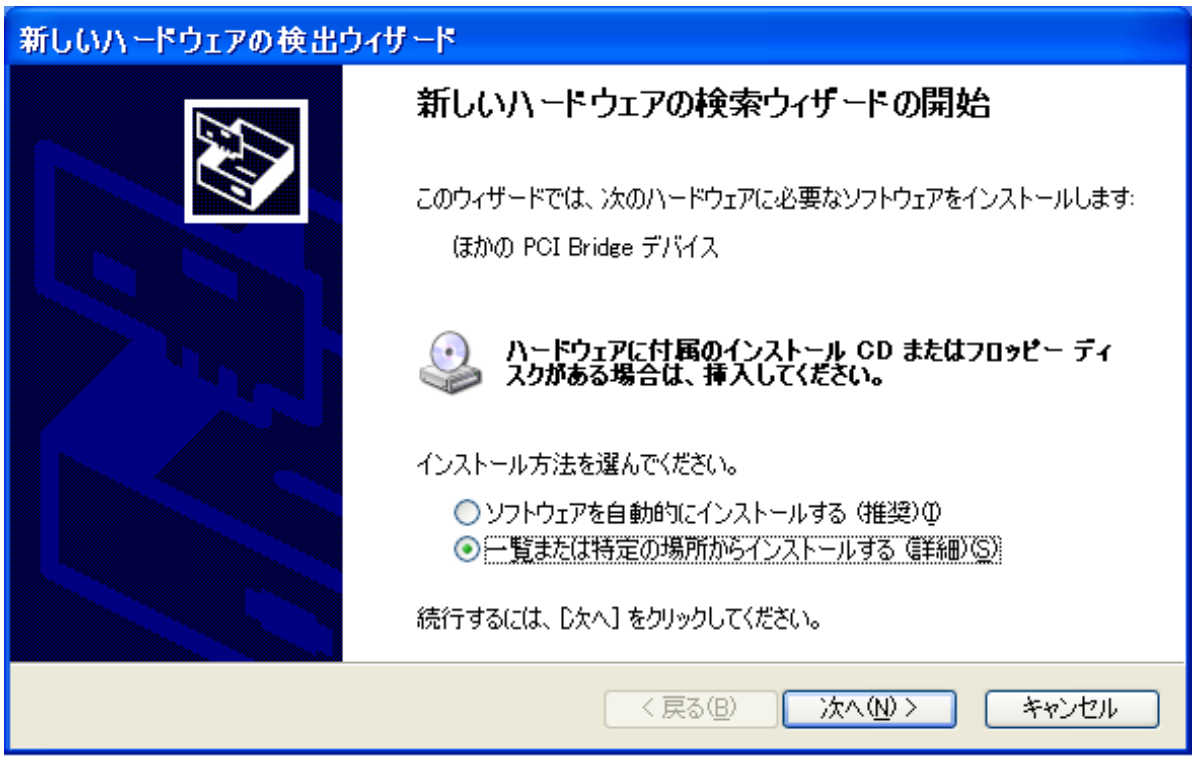
デバイスドライバのインストールは必ずアドミニストレーター権限をもったユーザーログインで行ってください。アドミニストレーター権限以外でインストールをした場合正常にインストールされません。

① 2.1の方法でインストールするデバイスドライバを準備して下さい。
② 2.2, 2.3 を実行し、本ボードが確実にパソコンに組み込まれているか確認してください。
③ パソコン本体の電源をＯＮし、Windows XPを起動します。
④ アドミニストレーター権限を持ったユーザーでログインしてください。

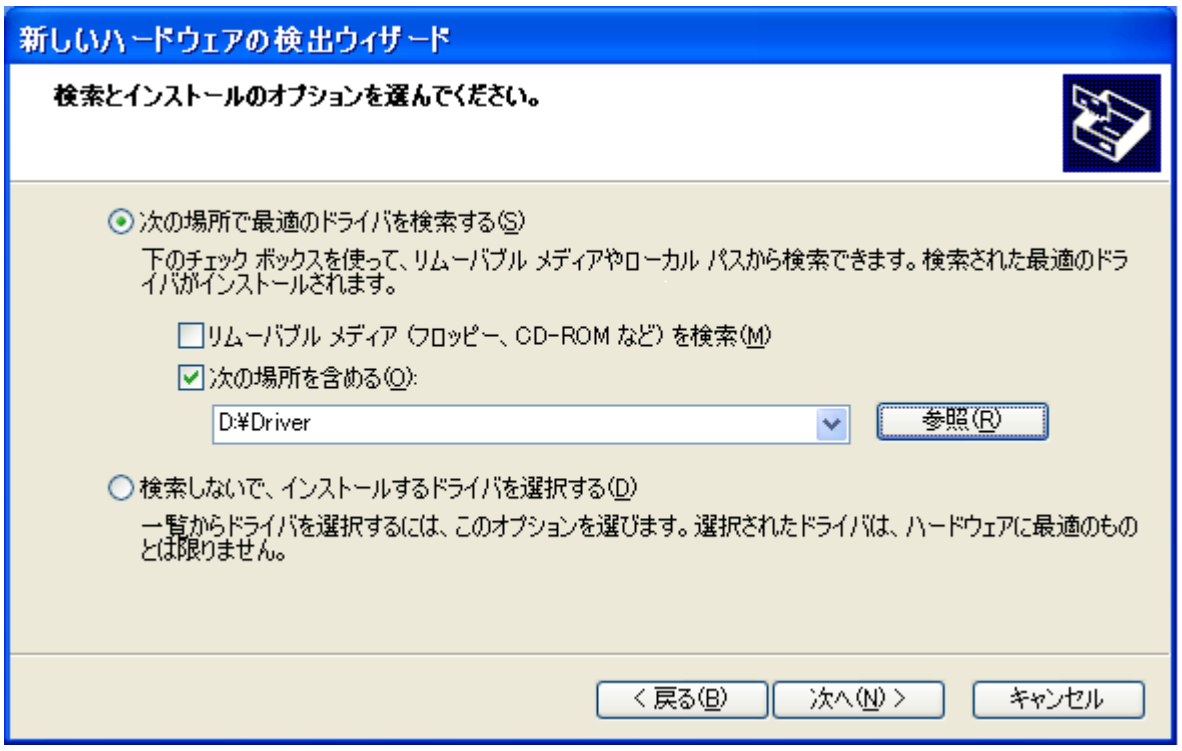
⑤ Windows XP サービスパック２の場合は、下の画面が表示されますので、「いいえ、今回は接続しません」を選択し、[次へ]をクリックします。
Windows XP サービスパック１の場合は、この画面は表示されませんので、次に進んで下さい。



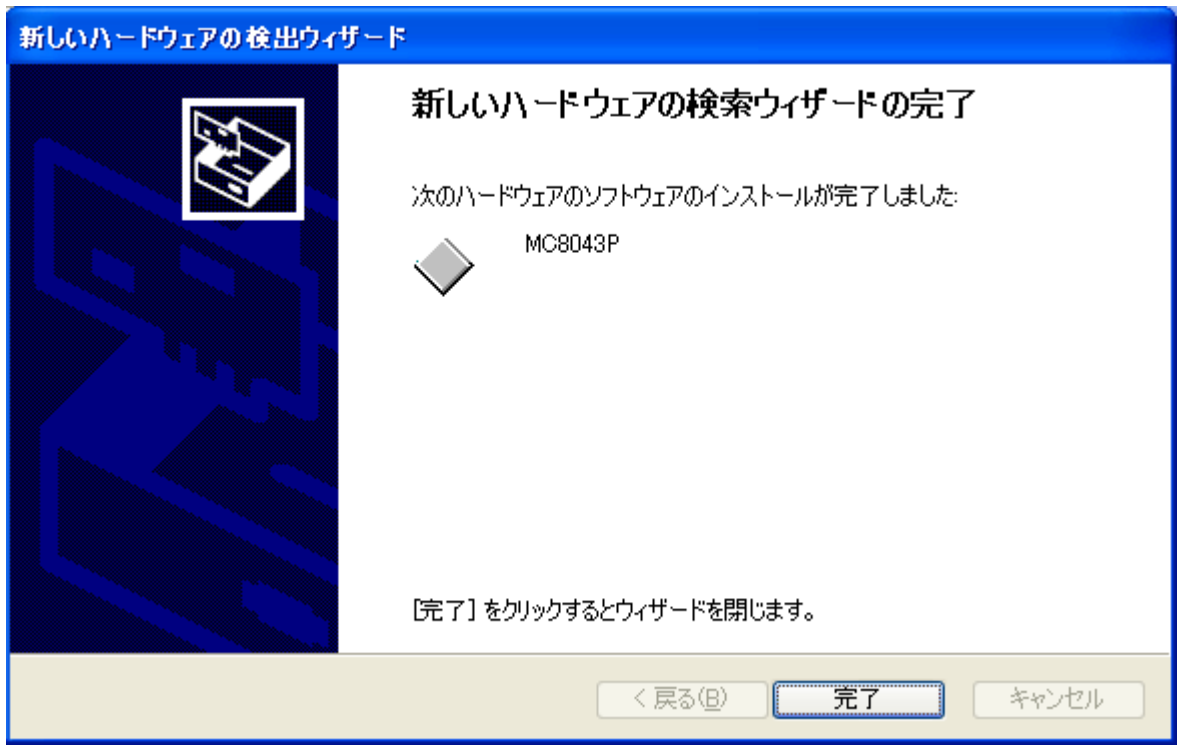
⑥ 次に下の画面が表示されますので「一覧または特定の場所からインストールする」を選択し、[次へ]をクリックします。



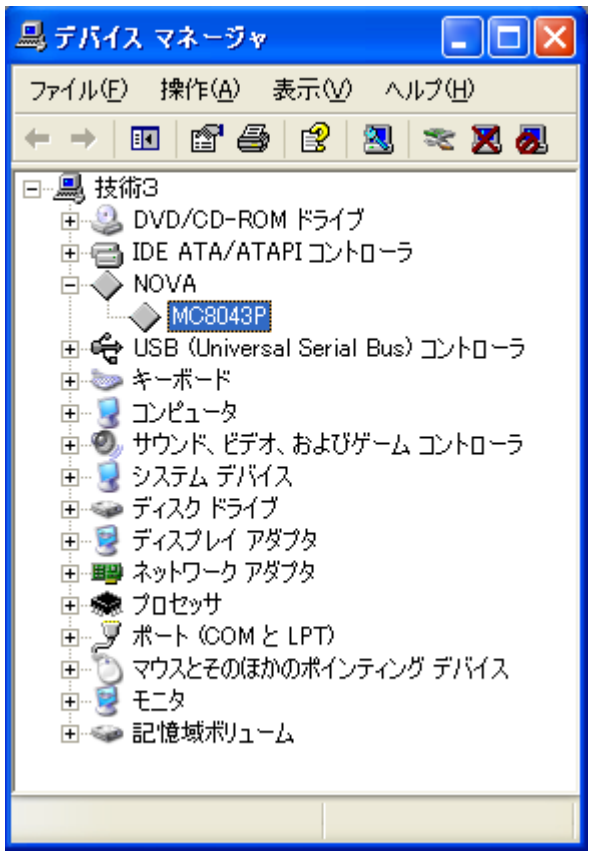
⑦　下の画面が表示されますので、「次の場所で最適のドライバを検索する」を選択し、「次の場所を含める」をチェックします。
ドライバをＣＤ－ＲＯＭからインストールする場合は、パソコンにＣＤ－ＲＯＭをセットし、ＣＤ－ＲＯＭが認識されてから次の動作に進みます。
参照ボタンを押し、提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、テキストボックスにフォルダが表示されたら、[次へ]をクリックします。

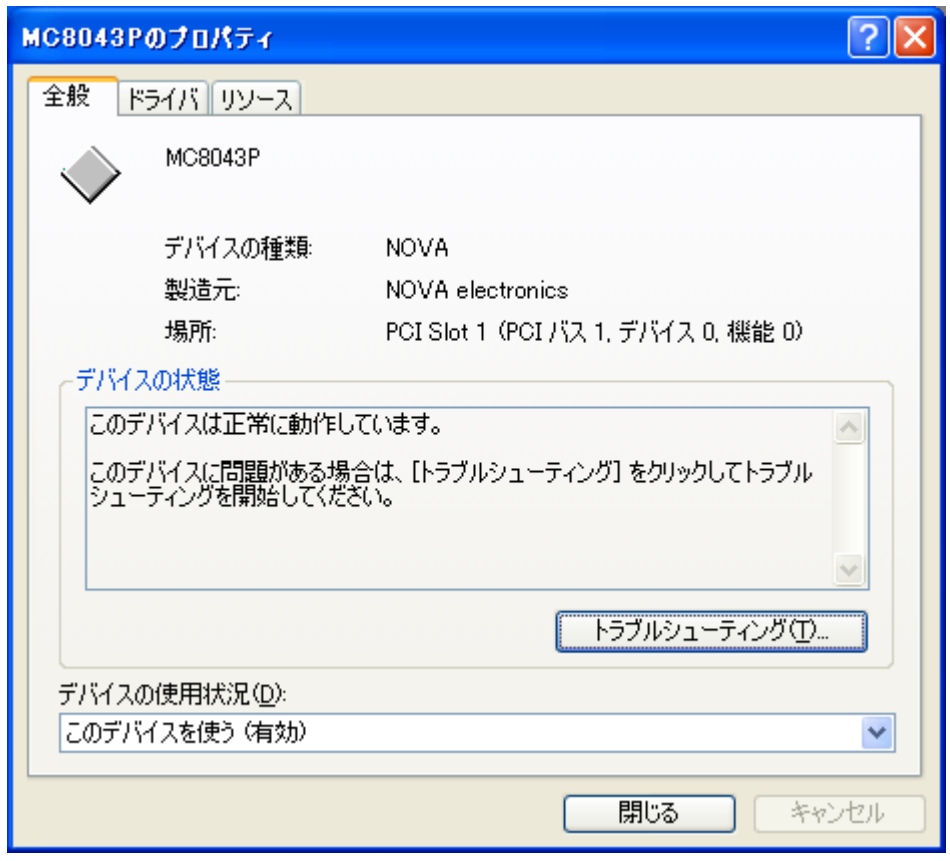


⑧　デバイスドライバのインストールが正常に完了すると、下の画面が表示されますので、[完了]をクリックします。



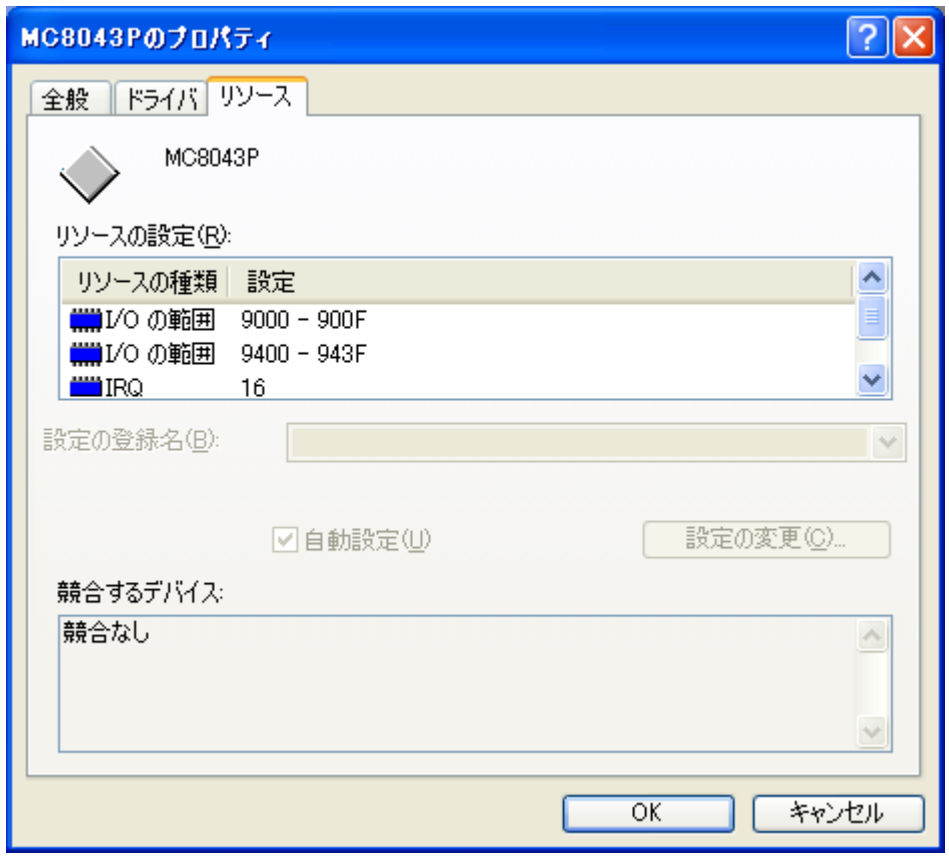
⑨　以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、
「ＮＯＶＡ」の下のボード名（ＭＣ８０４３Ｐなど）をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。





また、デバイスドライバのインストール完了後はパソコンの起動時に手順⑤、⑥のようにハードウェアウィザードが起動することはありません。もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので、その場合は2.5の手順に従って一度本ボードを取り外した後に2.3の手順から再度インストールをやり直してください。

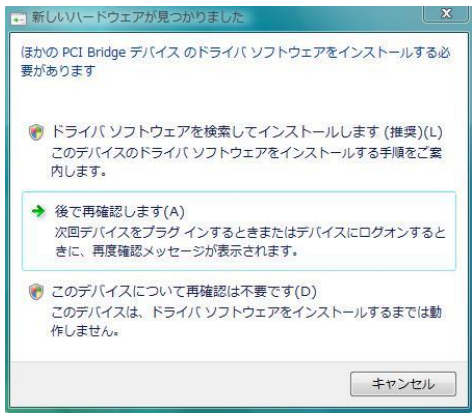
インストールを完了したらリソース(Ｉ／Ｏアドレス、割り込みレベル)の設定、競合の有無を[コントロールパネル]－[システム]－[ハードウェア]－[デバイスマネージャ]でボード名をダブルクリックし、[リソース]タブで確認してください。



2.4.4Windows Vista/7 (32bit)

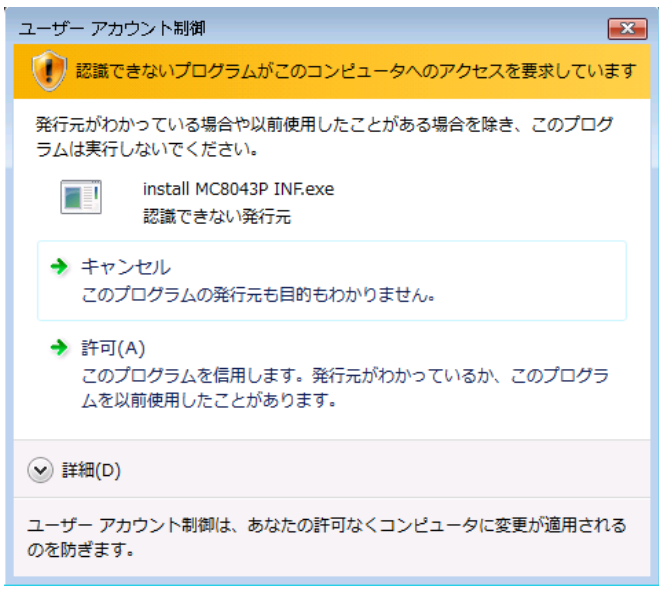
起動中の他のアプリケーションは終了させてください。
デバイスドライバのインストールは必ずアドミニストレーター権限をもったユーザーログインで行ってください。アドミニストレーター権限以外でインストールをした場合正常にインストールされません。
下記の手順はWindows Vistaを例にしていますが、Windows 7でもインストールの進め方は同じです。

① 2.1の方法でインストールするデバイスドライバを準備して下さい。
② 2.2, 2.3 を実行し、本ボードが確実にパソコンに組み込まれているか確認してください。
③ パソコン本体の電源をＯＮし、Windows Vistaを起動します。
④ アドミニストレーター権限を持ったユーザーでログインしてください。
⑤ Windows Vistaの場合は、下の画面が表示されますので、【後で再確認します（A）】をクリックします。 複数枚ボードがある場合はその数だけ表示されますので、同様に操作してください。

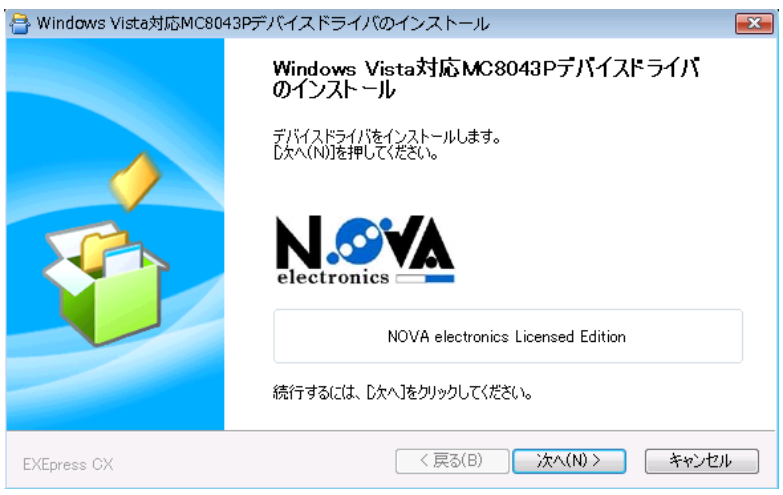


⑥ 提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、Vista Driverフォルダ内の「install xxxxx INF.exe」をダブルクリックしてください。（xxxxx にはお使いのモデル名が入ります。ＰＣＩ Ｅｘｐｒｅｓｓボードは、ＰＣＩボードと同じファイルを使用してください。例えば、ＭＣ８０４３Ｐｅは、「install MC8043P INF.exe」を使用します）

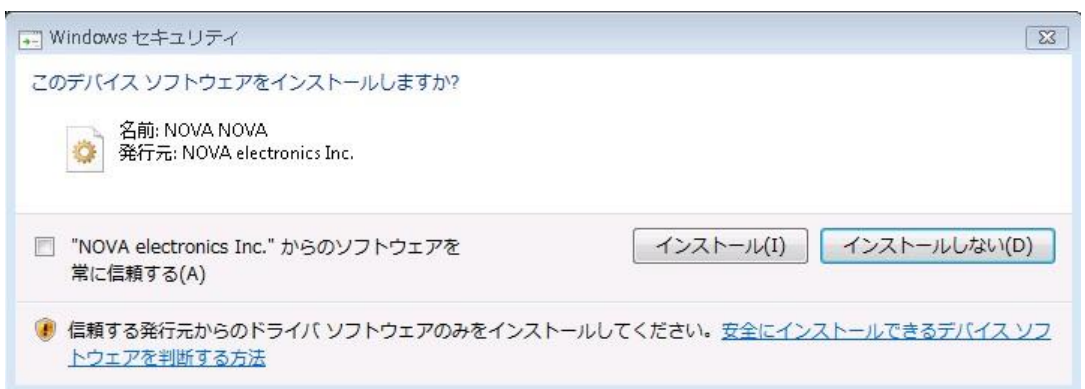
⑦ ”ユーザーアカウント制御” の画面が出てきた場合は、【許可】ボタンをクリックして操作を進めてください。（画面上のＭＣ８０４３Ｐはお使いのモデル名に置き換えてご覧ください。）



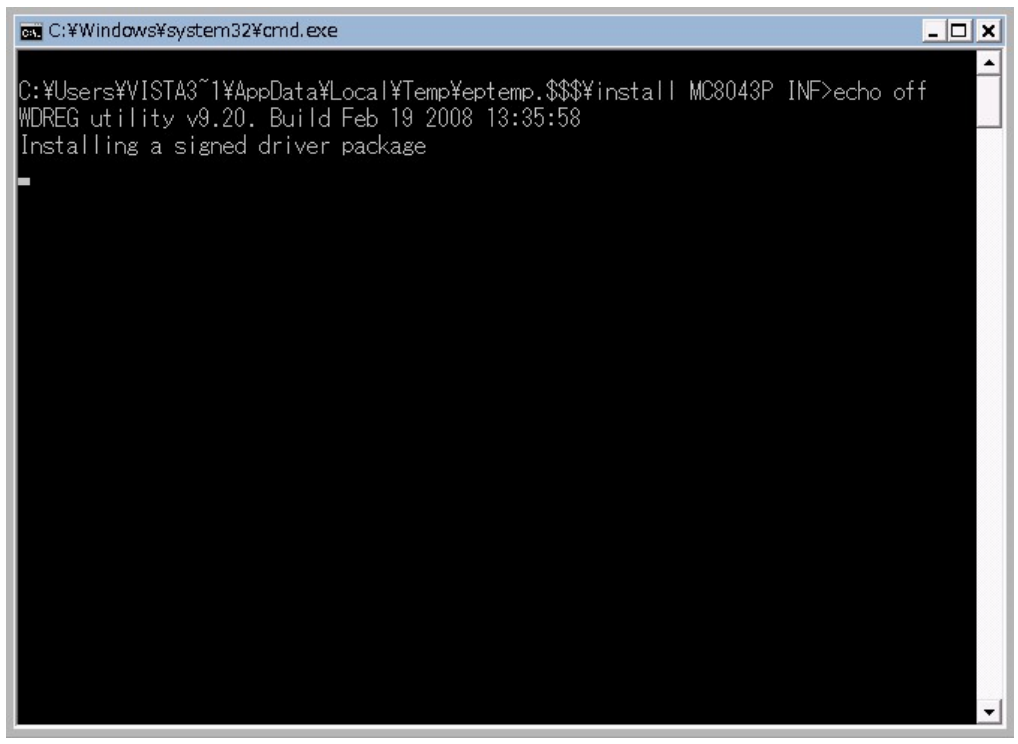
⑧　デバイスドライバのインストール画面
　　【次へ】のボタンをクリック（画面上のＭＣ８０４３Ｐはお使いのモデル名に置き換えてご覧ください。）



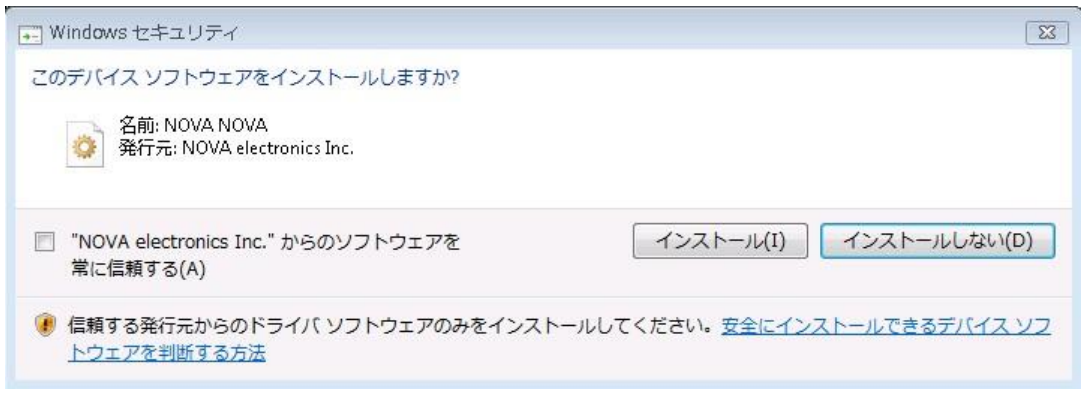
⑨　【インストール】をクリックします。



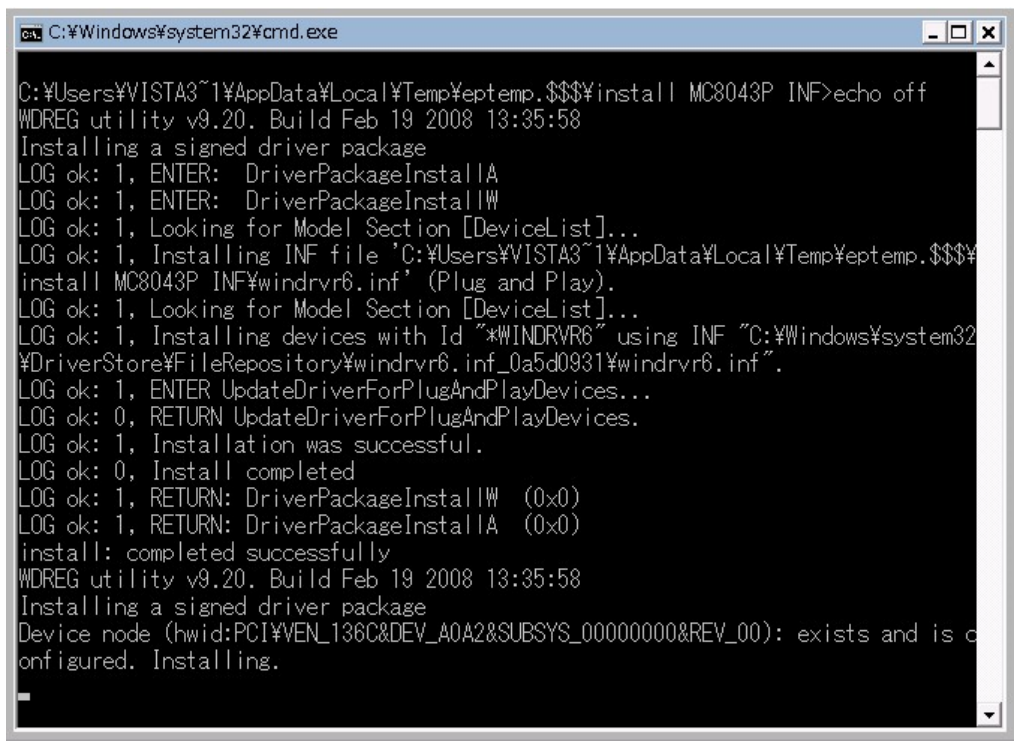
⑩　処理が終了すると自動的に画面が閉じられます。（インストールに時間がかかることがあります。完了のメッセージがあるまでお待ちください。）



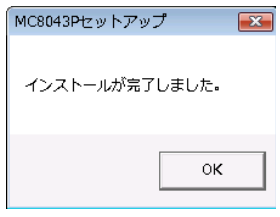
⑪　再度表示されることがあります。【インストール】をクリックします。



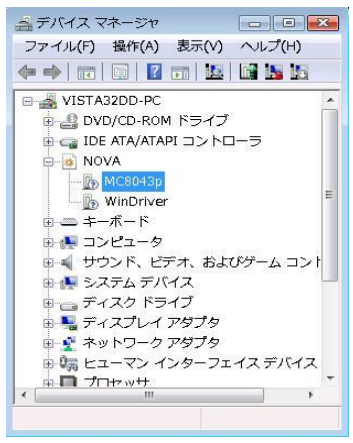
⑫　処理が終了すると自動的に画面が閉じられます。（インストールに時間がかかることがあります。完了のメッセージがあるまでお待ちください。）

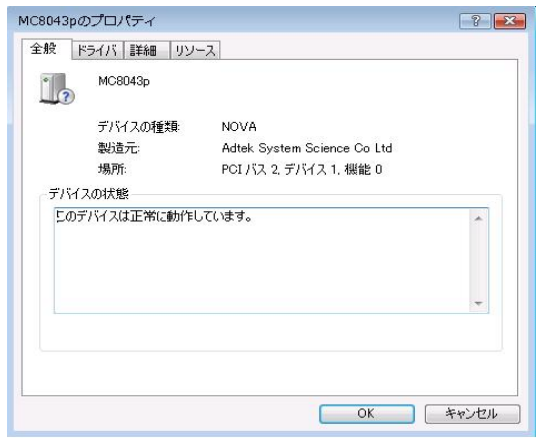


⑬　【ＯＫ】ボタンをクリックしてデバイスドライバのインストールは完了です。



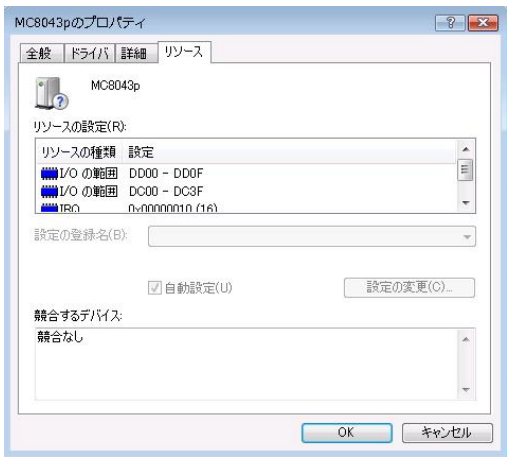
⑭　以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システムとメンテナンス」－「デバイスマネージャ」画面（下記左画面）を開き、「ＮＯＶＡ」の下のボード名（ＭＣxxxxxｐなど）をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。
（xxxxx にはお使いのモデル名が入ります）
（画面上のMC8043Pはお使いのモデル名に置き換えてご覧ください）
（ＰＣＩ　Ｅｘｐｒｅｓｓボードでも、ボード名表示はＰＣＩボードと同じです。
　例えば、ＭＣ８０４３Ｐｅをインストールした場合、ボード名表示は、ＭＣ８０４３Ｐとなります）





また、デバイスドライバのインストール完了後はパソコンの起動時に手順⑤のようにハードウェアウィザードが起動することはありません。もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので、その場合は⑥の手順から再度インストールをやり直してください。

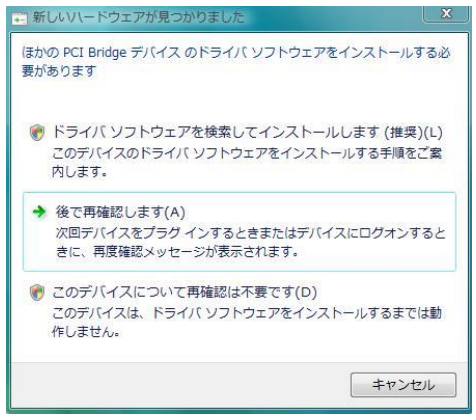
インストールを完了したらリソース（Ｉ／Ｏアドレス、割り込みレベル）の設定、競合の有無を［コントロールパネル］－［システムとメンテナンス］－［デバイスマネージャ］でボード名をダブルクリックし、［リソース］タブで確認してください。



2.4.5Windows Vista/7 (64bit)

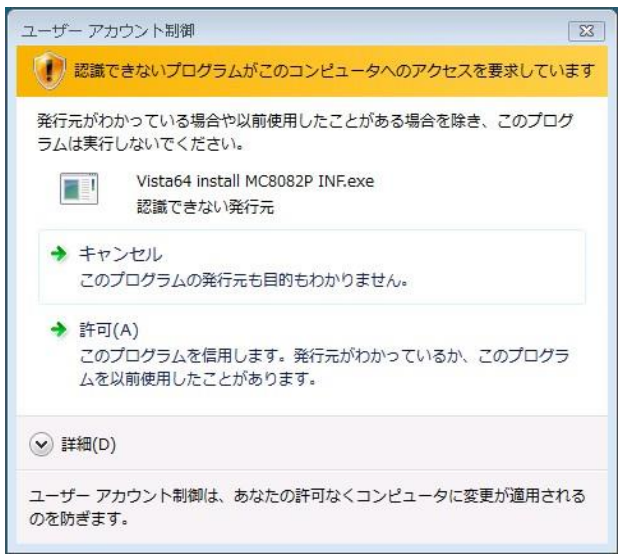
起動中の他のアプリケーションは終了させてください。
デバイスドライバのインストールは必ずアドミニストレーター権限をもったユーザーログインで行ってください。アドミニストレーター権限以外でインストールをした場合正常にインストールされません。
下記の手順はWindows Vistaを例にしていますが、Windows 7でもインストールの進め方は同じです。

① 2.1の方法でインストールするデバイスドライバを準備して下さい。
② 2.2, 2.3 を実行し、本ボードが確実にパソコンに組み込まれているか確認してください。
③ パソコン本体の電源をＯＮし、Windows Vistaを起動します。
④ アドミニストレーター権限を持ったユーザーでログインしてください。
⑤ Windows Vistaの場合は、下の画面が表示されますので、【後で再確認します（A）】をクリックします。 複数枚ボードがある場合はその数だけ表示されますので、同様に操作してください。



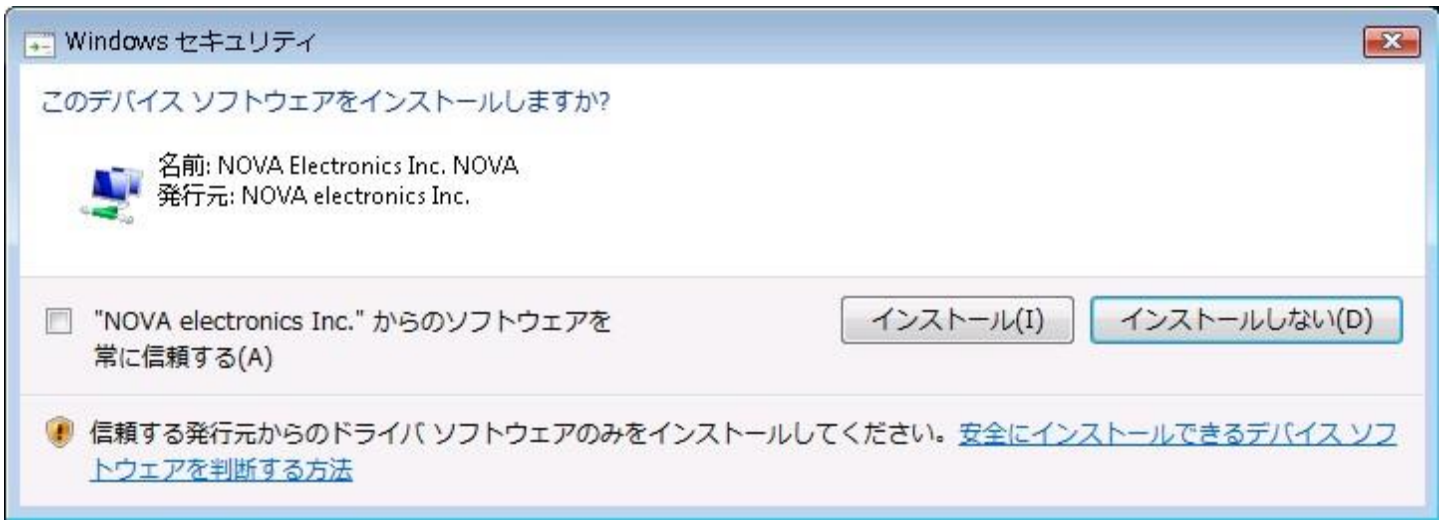
⑥ 提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、Vista Driverフォルダ内の「64bit install xxxxx INF.exe」をダブルクリックしてください。（xxxxx にはお使いのモデル名が入ります。ＰＣＩ Ｅｘｐｒｅｓｓボードは、ＰＣＩボードと同じファイルを使用してください。例えば、ＭＣ８０８２Ｐｅは、「64bit install MC8082P INF.exe」を使用します）

⑦ ”ユーザーアカウント制御” の画面が出てきた場合は、【許可】ボタンをクリックして操作を進めてください。（画面上のＭＣ８０８２Ｐはお使いのモデル名に置き換えてご覧ください。）

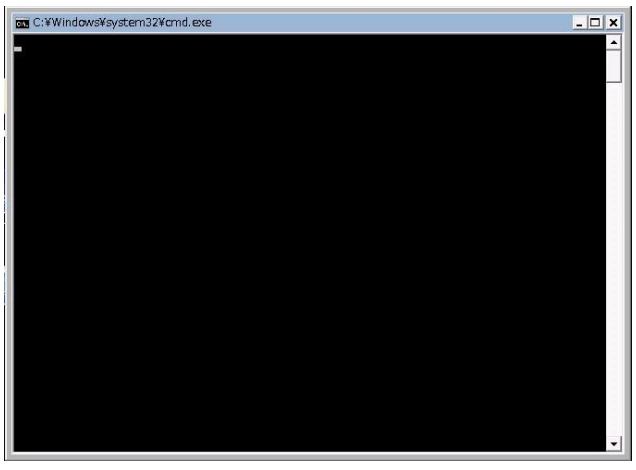


⑧　デバイスドライバのインストール画面
　　【次へ】のボタンをクリック
　　（画面上のＭＣ８０８２Ｐはお使いのモデル名に置き換えてご覧ください。）

⑨　【インストール】をクリックします。

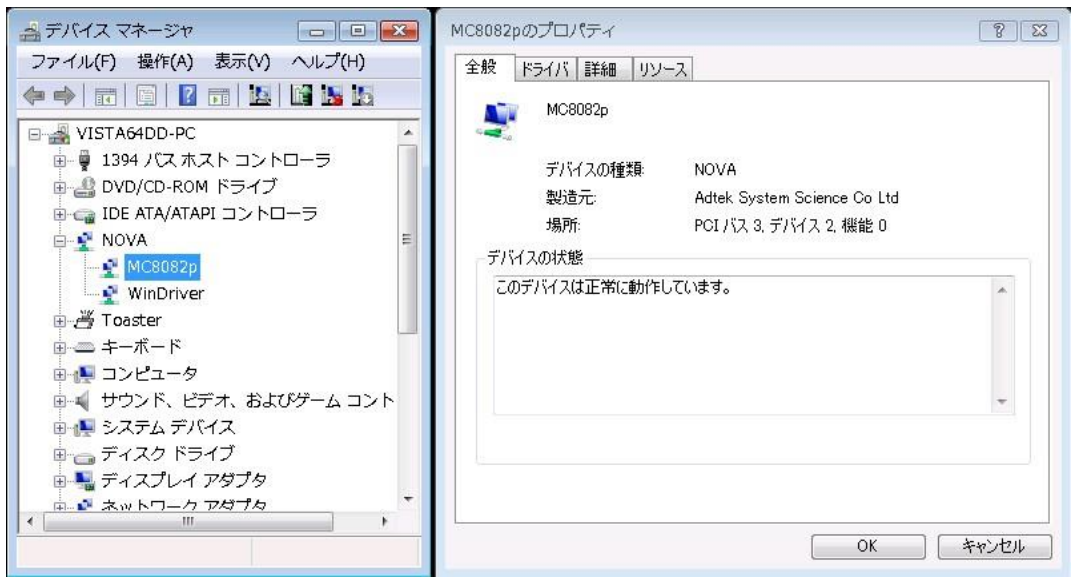


⑩　処理が終了すると自動的に画面が閉じられます。（インストールに時間がかかることがあります。完了のメッセージがあるまでお待ちください。）



⑪　【ＯＫ】ボタンをクリックしてデバイスドライバのインストールは完了です。

⑫　以上でデバイスドライバのインストールは完了です。
次の方法で正しくインストールされたかどうかを確認して下さい。
「コントロールパネル」－「システムとメンテナンス」－「デバイスマネージャ」画面（下記左画面）を開き、「ＮＯＶＡ」の下のボード名（ＭＣxxxxxｐなど）をダブルクリックし、「全般」タブで下記右画面を表示します。
この画面でデバイスの状態に「このデバイスは正常に動作しています」と記載されていたらインストールは正常終了です。
（xxxxx にはお使いのモデル名が入ります）
（画面上のＭＣ８０８２Ｐはお使いのモデル名に置き換えてご覧ください）
（ＰＣＩ　Ｅｘｐｒｅｓｓボードでも、ボード名表示はＰＣＩボードと同じです。例えば、ＭＣ８０８２Ｐｅをインストールした場合、　ボード名表示は、ＭＣ８０８２Ｐとなります）



通常、デバイスドライバのインストール完了後はパソコンの起動時に手順⑤のようにハードウェアウィザードが起動することはありません。もしハードウェアウィザードが起動するような場合はインストールが正常に終了していない可能性がありますので、再度⑥の手順から再度インストールをやり直してください。

インストールを完了したらリソース(Ｉ／Ｏアドレス、割り込みレベル)の設定、競合の有無を[コントロールパネル]－[システムとメンテナンス]－[デバイスマネージャ]でボード名をダブルクリックし、[リソース]タブで確認してください。

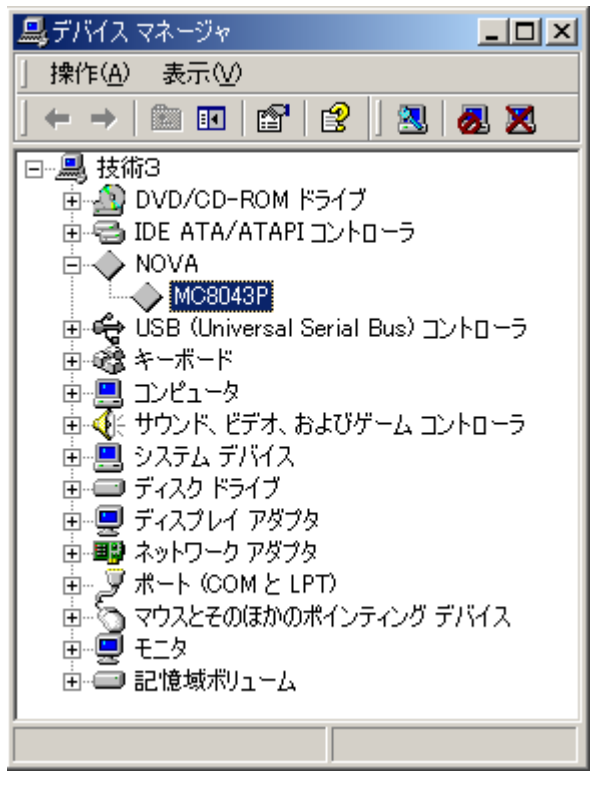
2.5　取り外し

2.5.1Windows 98 (32bit)

①まず[コントロールパネル]-[システム]-[デバイスマネージャ]タブで本製品のデバイスドライバを削除してください。
②パソコン本体の電源がＯＦＦであることを確認してから、外装カバー､スロットカバー等を外します。
③ボードを止めているビスを外します。
④本ボードを指先でつまんで軽く左右にゆするようにしながら引き出します。
⑤パソコン本体の電源をＯＮし、Windows 98を起動します。
⑥[コントロールパネル]-[システム]-[デバイスマネージャ]タブで本製品が削除されていることを確認してください。

2.5.2Windows 2000/XP (32bit)

①　まず[コントロールパネル]-[システム]-[ハードウェア]-[デバイスマネージャ]で本製品のデバイスドライバを削除してください。
②　パソコン本体の電源がＯＦＦであることを確認してから、外装カバー､スロットカバー等を外します。
③　ボードを止めているビスを外します。
④　本ボードを指先でつまんで軽く左右にゆするようにしながら引き出します。
⑤　パソコン本体の電源をＯＮし、Windows 2000/XPを起動します。
⑥　[コントロールパネル]-[システム]-[ハードウェア]-[デバイスマネージャ]タブで本製品が削除されていることを確認して下さい。

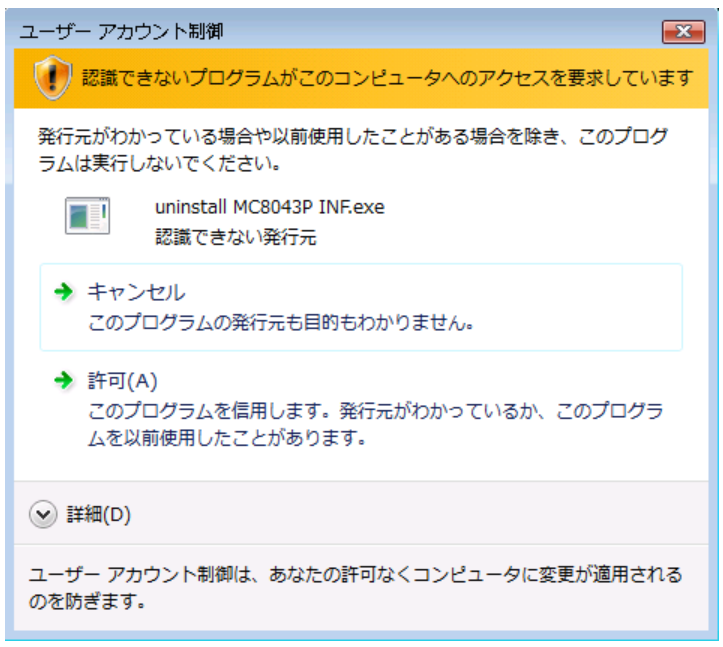


2.5.3Windows Vista/7 (32bit)

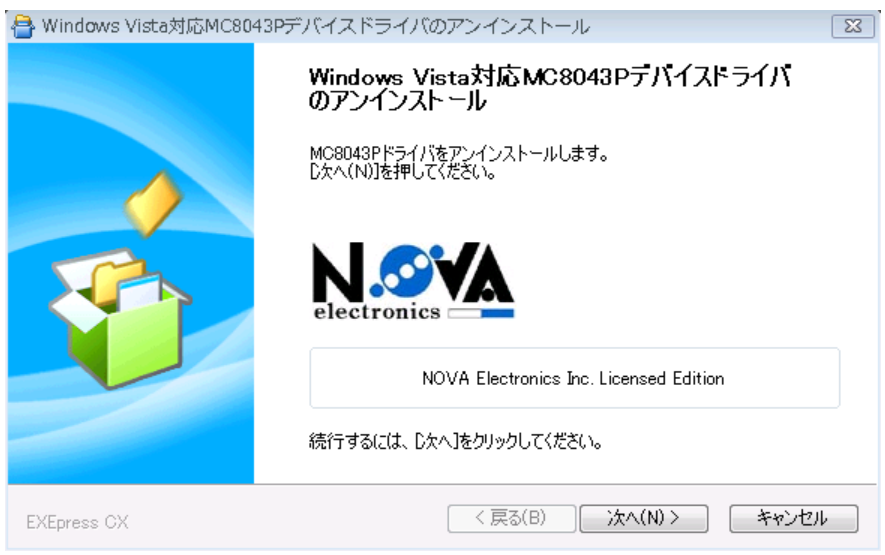
ボードをPCから外す場合、外す前に必ず下記の手順にそって、デバイスドライバを削除してください。

① 提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、Vista Driverフォルダ内の「uninstall xxxxx INF.exe」をダブルクリックしてください。
下記の手順はWindows Vistaを例にしていますが、Windows 7でもアンインストールの進め方は同じです。
（xxxxx にはお使いのモデル名が入ります。ＰＣＩ Ｅｘｐｒｅｓｓボードは、ＰＣＩボードと同じファイルを使用してください。例えば、ＭＣ８０４３Ｐｅは、「uninstall MC8043P INF.exe」を使用します）
（画面上のＭＣ８０４３Ｐはお使いのモデル名に置き換えてご覧ください。）

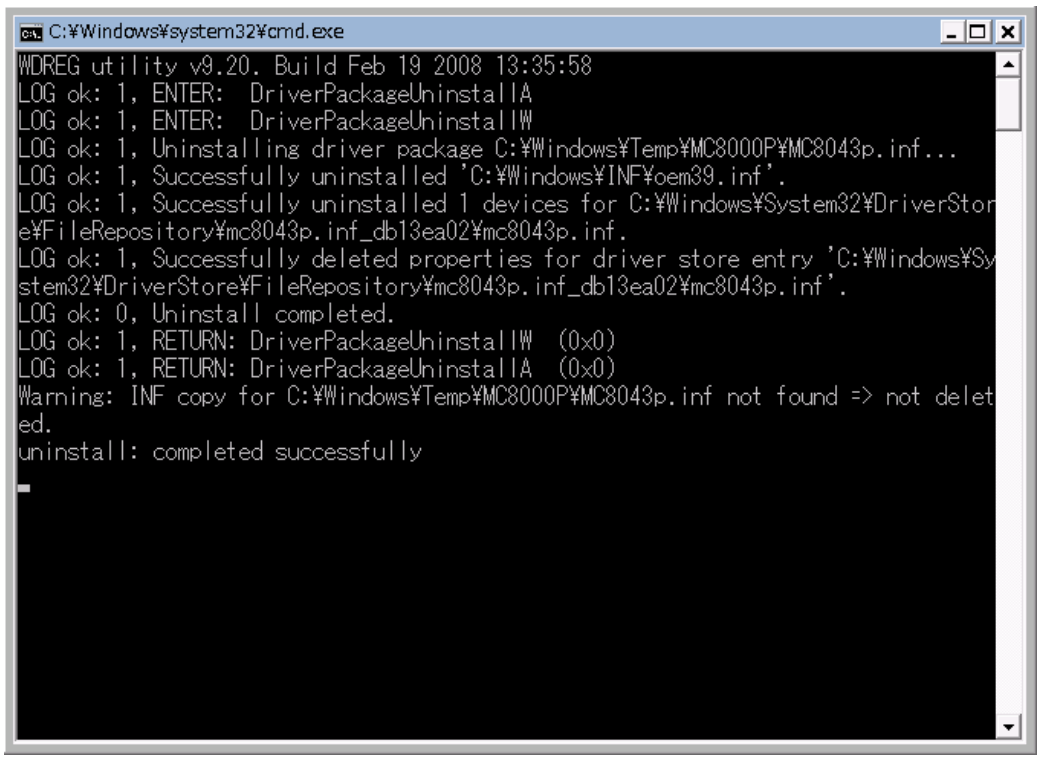
② ”ユーザーアカウント制御” の画面が出てきた場合は、【許可】ボタンをクリックして操作を進めてください。
（画面上のＭＣ８０４３Ｐはお使いのモデル名に置き換えてご覧ください。）



③ デバイスドライバのアンインストール画面
【次へ】のボタンをクリック（画面上のＭＣ８０４３Ｐはお使いのモデル名に置き換えてご覧ください。）

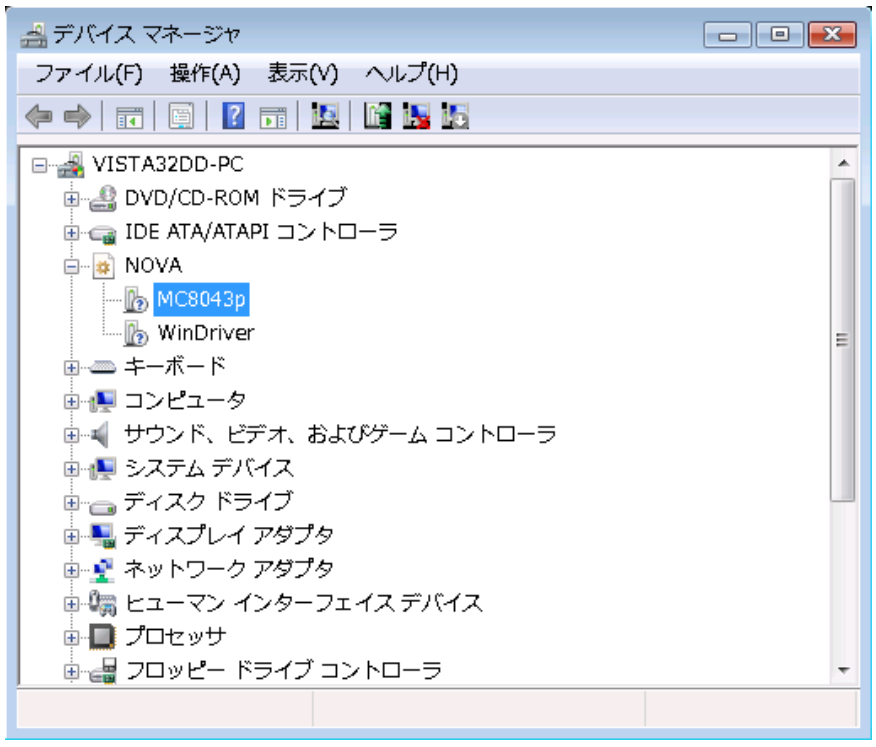


④　処理が終了すると自動的に画面が閉じられます。（アンインストールに時間がかかることがあります。完了のメッセージがあるまでお待ちください。）



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

⑥　パソコン本体の電源がＯＦＦであることを確認してから、外装カバー､スロットカバー等を外します。
⑦　ボードを止めているビスを外します。
⑧　本ボードを指先でつまんで軽く左右にゆするようにしながら引き出します。
⑨　パソコン本体の電源をＯＮし、Windows Vistaを起動します。
⑩　[コントロールパネル]-[システム]-[ハードウェア]-[デバイスマネージャ]タブで本製品が削除されていることを確認して下さい。
取り外すボードがＭＣ８０４３Ｐとすると、図中の反転部分（ＭＣ８０４３Ｐ）が削除されていることを確認してください。

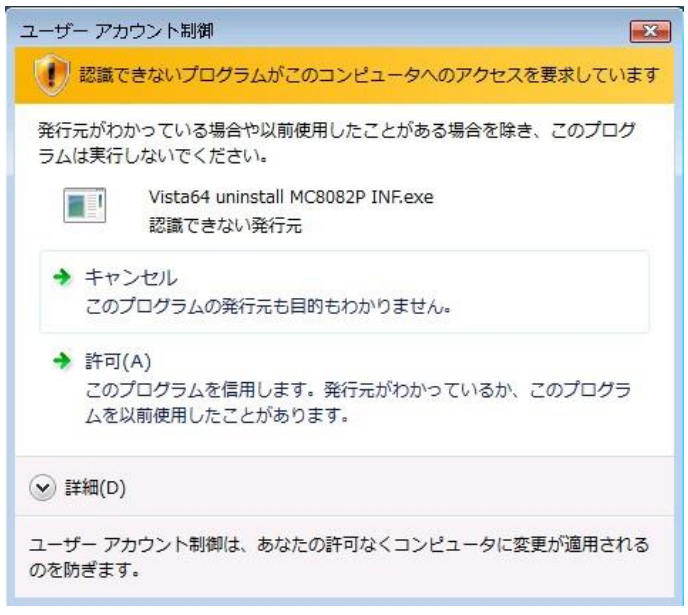


2.5.4Windows Vista/7 (64bit)

ボードをPCから外す場合、外す前に必ず下記の手順にそって、デバイスドライバを削除してください。

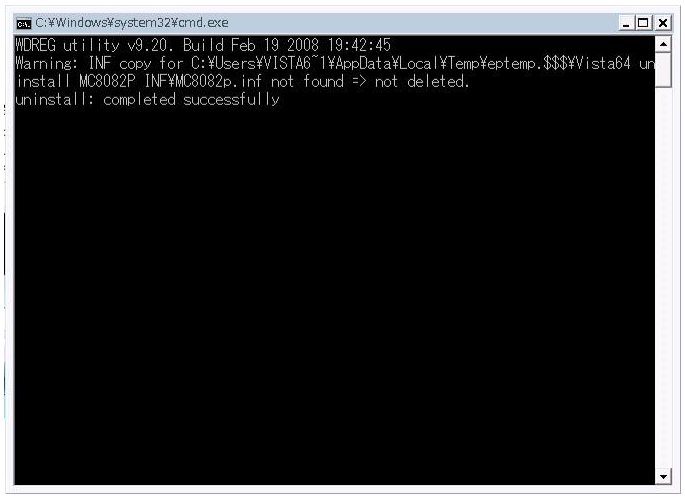
① 提供ＣＤ－ＲＯＭのDriverフォルダ（提供ＣＤ－ＲＯＭがＤドライブにある場合は、D:¥Driver）、あるいはダウンロードしたソフトウェアのDriverフォルダを選択し、Vista Driverフォルダ内の「64bit uninstall xxxxx INF.exe」をダブルクリックしてください。
下記の手順はWindows Vistaを例にしていますが、Windows 7でもアンインストールの進め方は同じです。
（xxxxx にはお使いのモデル名が入ります。ＰＣＩ Ｅｘｐｒｅｓｓボードは、ＰＣＩボードと同じファイルを使用してください。例えば、ＭＣ８０８２Ｐｅは、「64bit uninstall MC8082P INF.exe」を使用します）
（画面上のＭＣ８０８２Ｐはお使いのモデル名に置き換えてご覧ください。）

② ”ユーザーアカウント制御” の画面が出てきた場合は、【許可】ボタンをクリックして操作を進めてください。
（画面上のＭＣ８０８２Ｐはお使いのモデル名に置き換えてご覧ください。）



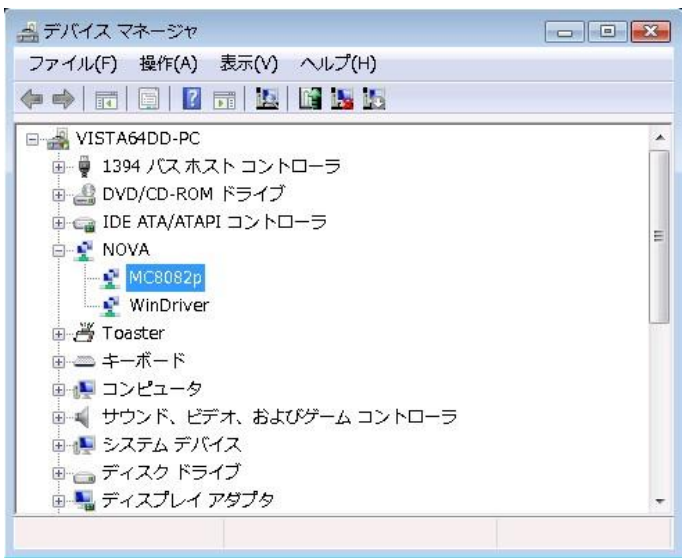
③ デバイスドライバのアンインストール画面
【次へ】のボタンをクリック（画面上のＭＣ８０８２Ｐはお使いのモデル名に置き換えてご覧ください。）

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



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

⑥　パソコン本体の電源がＯＦＦであることを確認してから、外装カバー､スロットカバー等を外します。
⑦　ボードを止めているビスを外します。
⑧　本ボードを指先でつまんで軽く左右にゆするようにしながら引き出します。
⑨　パソコン本体の電源をＯＮし、Windows Vistaを起動します。
⑩　取り外すボードがＭＣ８０８２Ｐとすると、図中の反転部分（ＭＣ８０８２Ｐ）が削除されていることを確認してください。

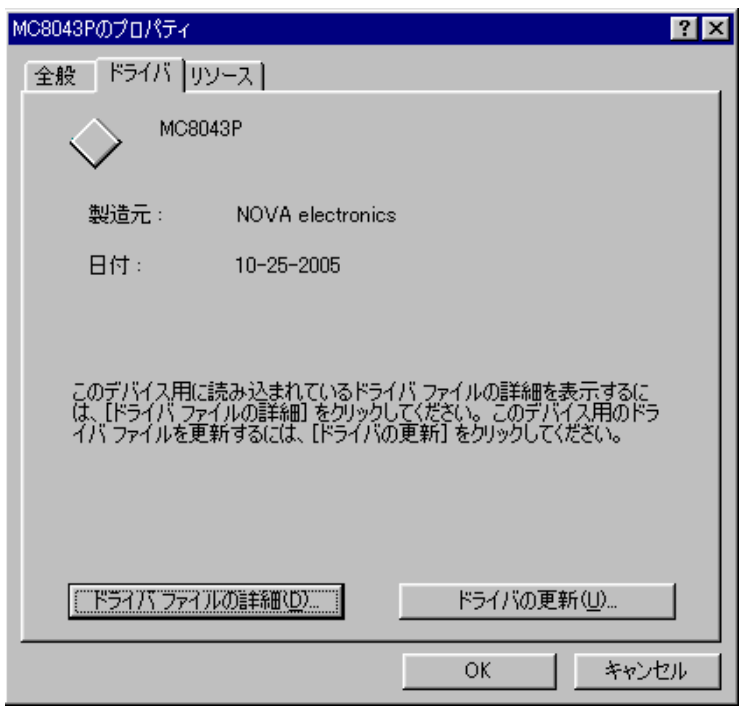


2.6　デバイスドライバの更新

デバイスドライバのバージョンが新しくなった場合、次の手順でドライバの更新を行って下さい。
以下に、各ＯＳに対応した更新手順を説明します。

2.6.1Windows 98 (32bit)

① 「コントロールパネル」－「システム」－「デバイスマネージャ」画面（下記左画面）を開き、「ＮＯＶＡ」の下のボード名（ＭＣ８０４３Ｐなど）をダブルクリックし、「ドライバ」タブで下記右画面を表示します。
② 次に「ドライバの更新」ボタンをクリックし、その次の画面で「次へ」ボタンをクリックします。



③下記画面が表示されますので、「特定の場所にあるすべてのドライバの一覧を作成し、インストールするドライバを選択する」を選択し、「次へ」ボタンをクリックします。

④下記画面で、「ディスク使用」ボタンをクリックし、「参照」ボタンから更新するドライバソフトのフォルダ（¥Driver）を選択後、下記画面の「次へ」ボタンをクリックします。その次の画面でも「次へ」ボタンをクリックすると、ドライバが更新されます。

⑤正常に更新されると、下記画面が表示されます。

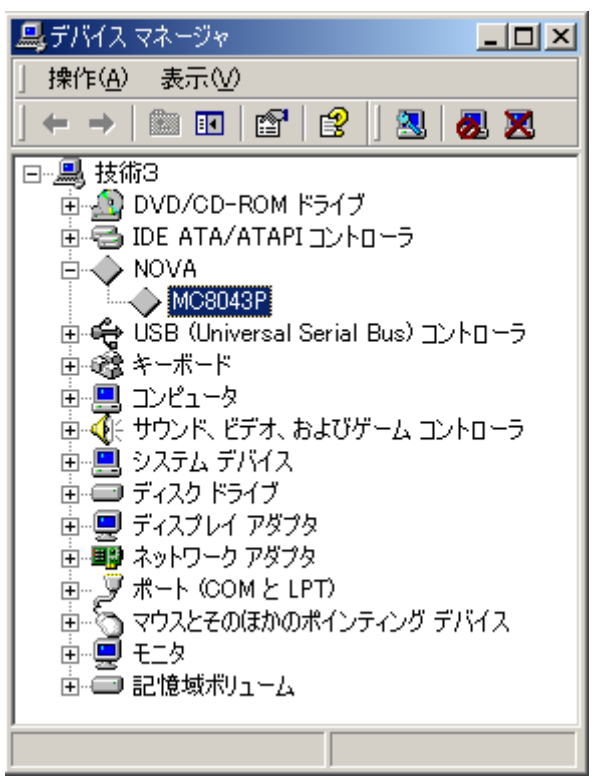
⑥「コントロールパネル」－「システム」－「デバイスマネージャ」でボード名をダブルクリックし、「ドライバ」タブで「ドライバの詳細」ボタンをクリックすると、下記画面が表示されます。
下記画面で、更新したドライバのファイルバージョンを確認します。
¥Driver¥Version.txt ファイルの「２．ドライバファイルバージョン」のバージョンが下記画面に表示されます。
Version.txt ファイルに記載している２つのファイルのバージョンが正しいか確認して下さい。

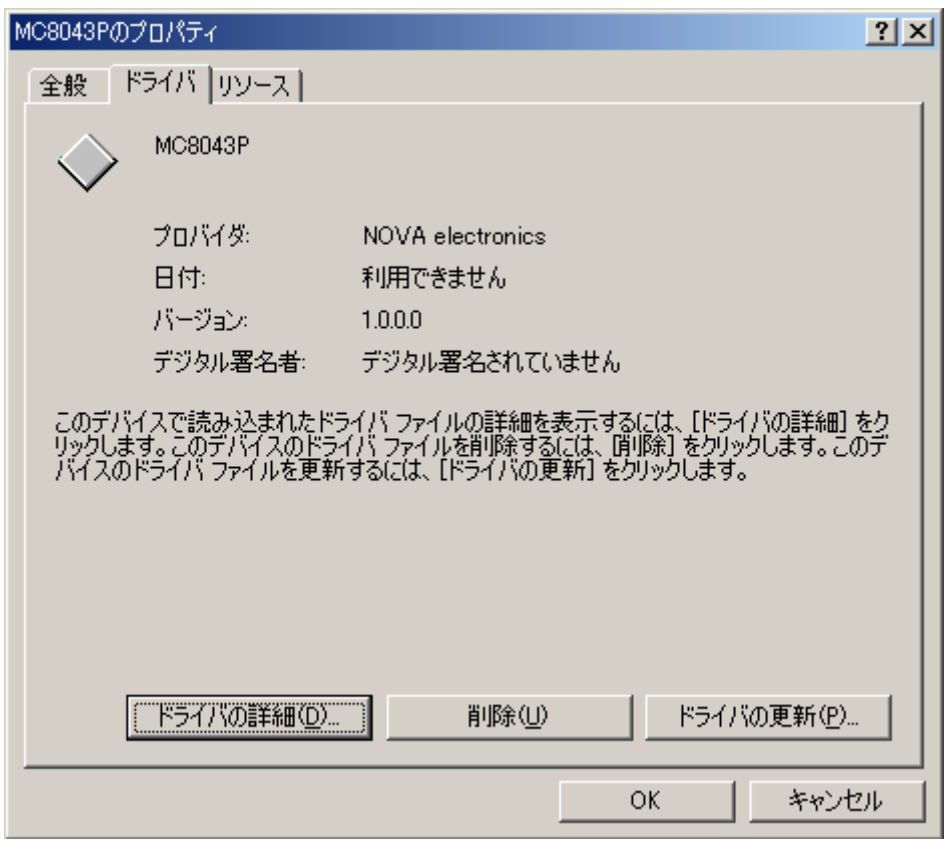
⑦ボードが複数ある場合は、「デバイスマネージャ」画面の「ＮＯＶＡ」の下に表示される全てのボードについて、①からの手順でドライバを更新して下さい。

⑧パソコンを再起動して下さい。これで更新作業は終了です。

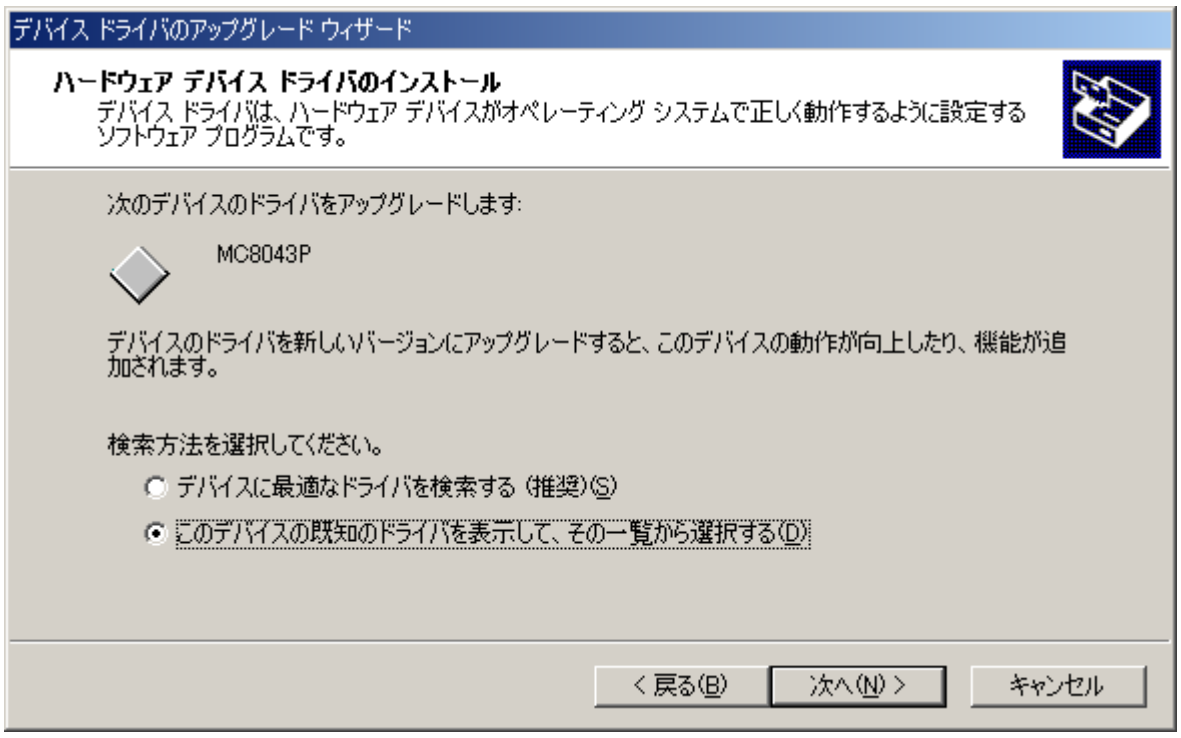
2.6.2Windows 2000 (32bit)

①「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、
「ＮＯＶＡ」の下のボード名（ＭＣ８０４３Ｐなど）をダブルクリックし、「ドライバ」タブで下記右画面を表示します。
②次に「ドライバの更新」ボタンをクリックし、次に表示された画面では「次へ」ボタンをクリックします。

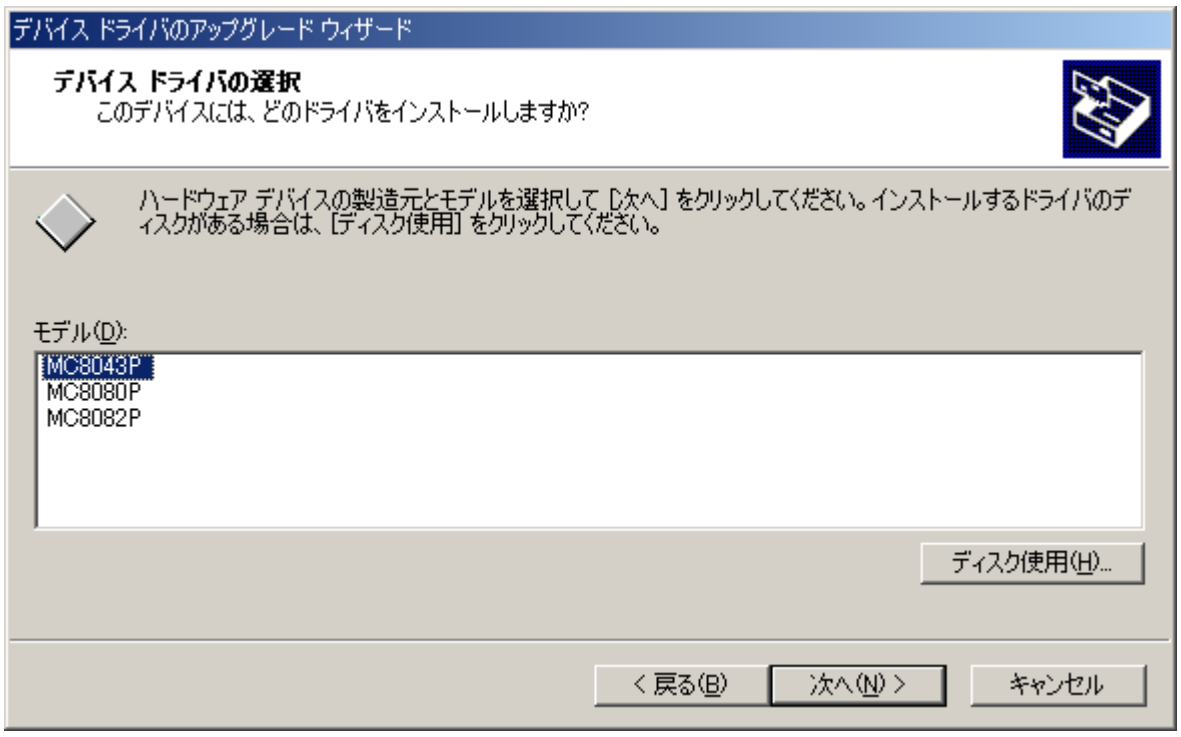




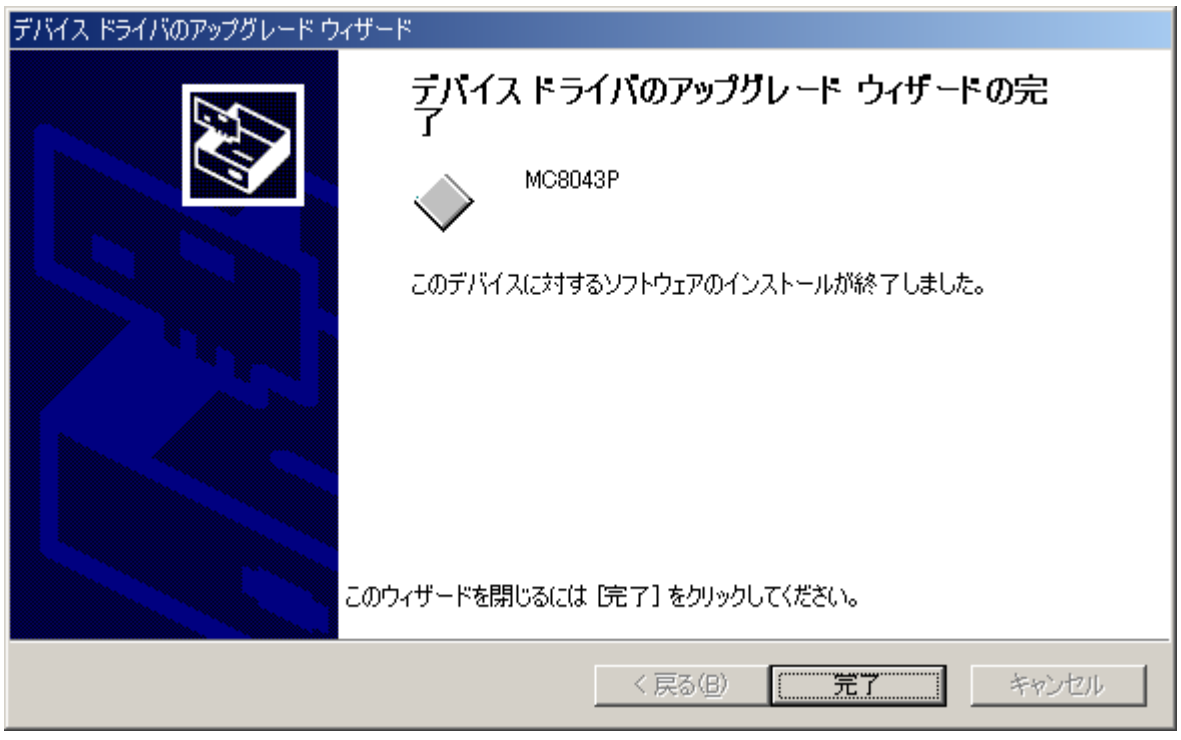
③下記画面が表示されますので、「このデバイスの既知のドライバを表示して、その一覧から選択する」を選択し、「次へ」ボタンをクリックします。



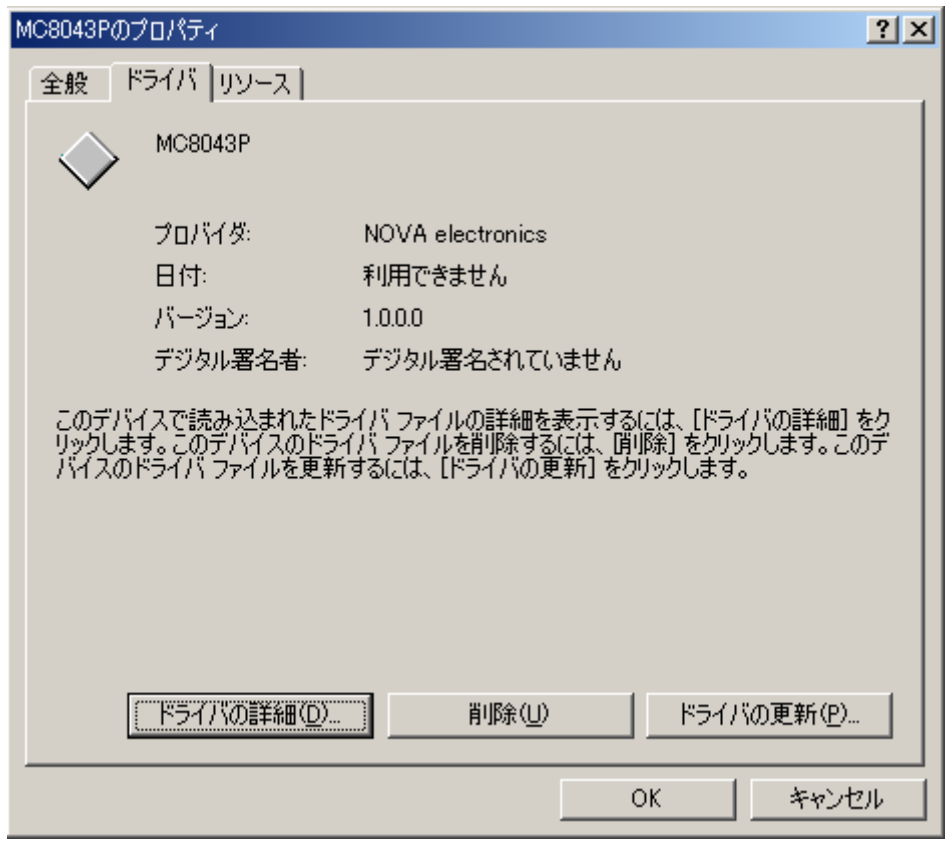
④下記画面で、「ディスク使用」ボタンをクリックし、「参照」ボタンから更新するドライバソフトのフォルダ（¥Driver）を選択後、下記画面のモデル一覧からボードを選択し「次へ」ボタンをクリックします。その次の画面でも「次へ」ボタンをクリックすると、ドライバが更新されます。



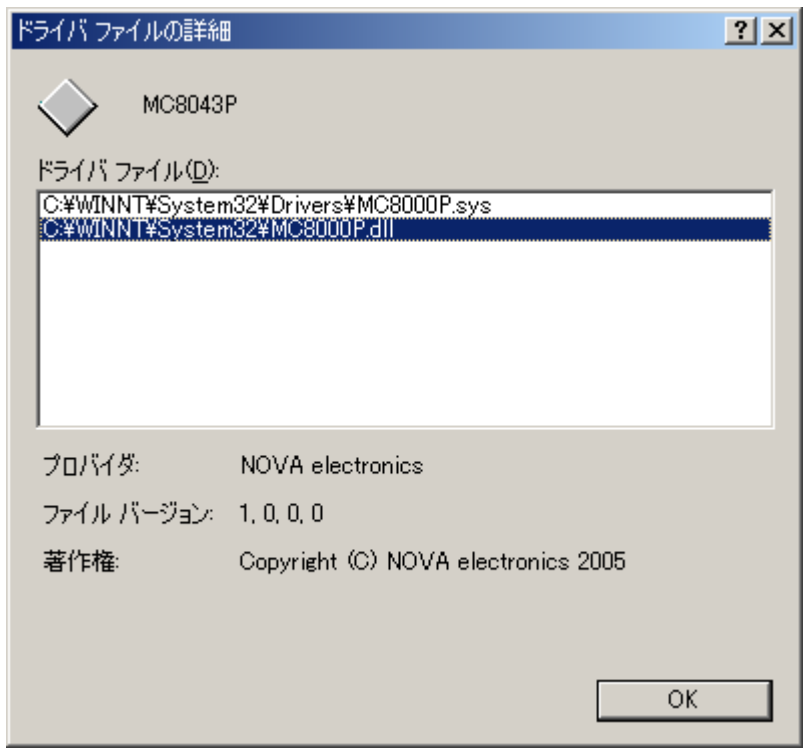
⑤正常に更新されると、下記画面が表示されます。



⑥下記画面（デバイスマネージャの更新したボードのプロパティ画面）で、更新したドライバのバージョンを確認します。
¥Driver¥Version.txt ファイルの「１．ドライババージョン」のバージョンが下記画面に表示されますので正しいか確認して下さい。 次に「ドライバの詳細」ボタンをクリックします。



⑦下記画面で、更新したドライバのファイルバージョンを確認します。
¥Driver¥Version.txt ファイルの「２．ドライバファイルバージョン」のバージョンが下記画面に表示されます。
Version.txt ファイルに記載している２つのファイルのバージョンが正しいか確認して下さい。

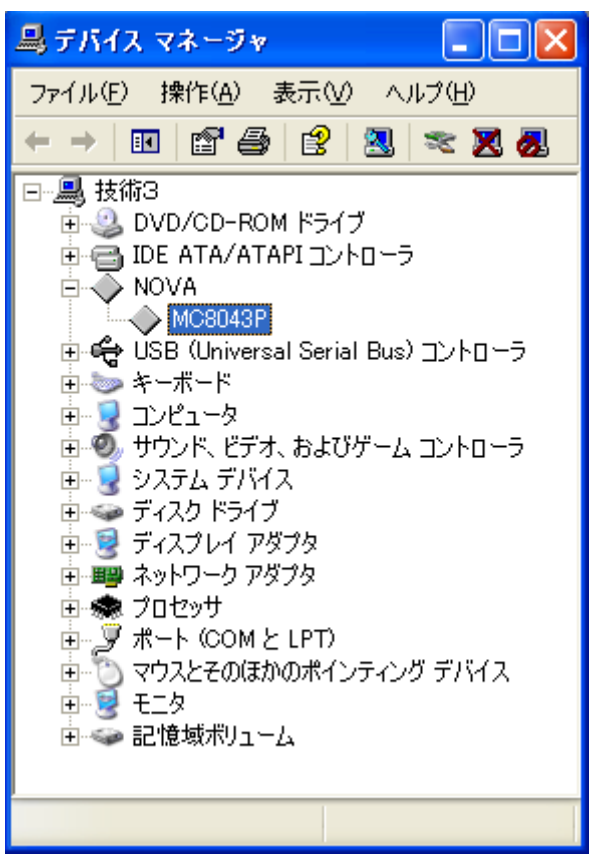


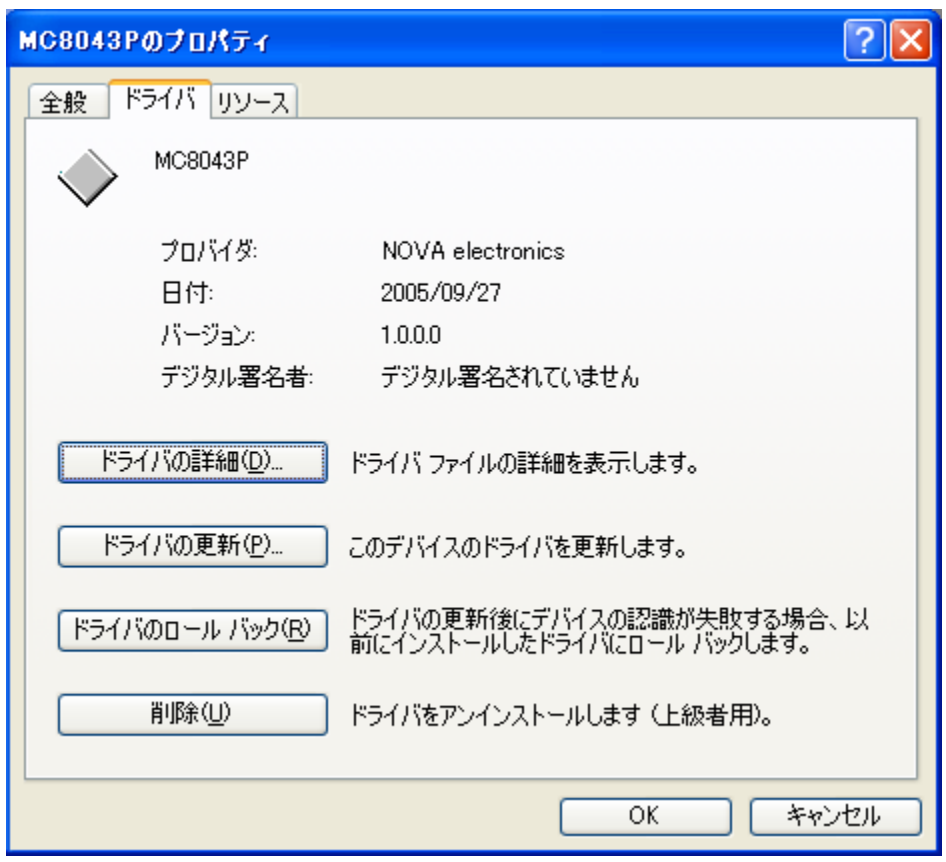
⑧ボードが複数ある場合は、「デバイスマネージャ」画面の「ＮＯＶＡ」の下に表示される全てのボードについて、①からの手順でドライバを更新して下さい。

⑨パソコンを再起動して下さい。これで更新作業は終了です。

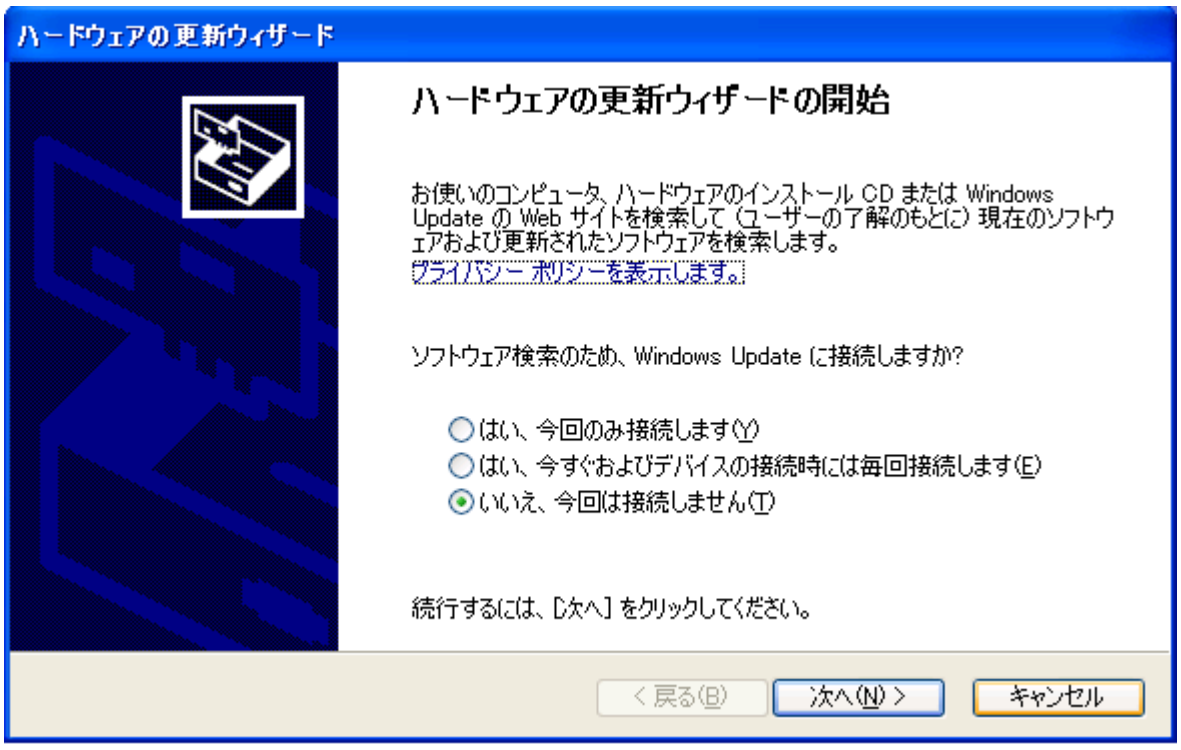
2.6.3 Windows XP (32bit)

①「コントロールパネル」－「システム」－「ハードウェア」－「デバイスマネージャ」画面（下記左画面）を開き、
「ＮＯＶＡ」の下のボード名（ＭＣ８０４３Ｐなど）をダブルクリックし、「ドライバ」タブで下記右画面を表示します。
②次に「ドライバの更新」ボタンをクリックします。

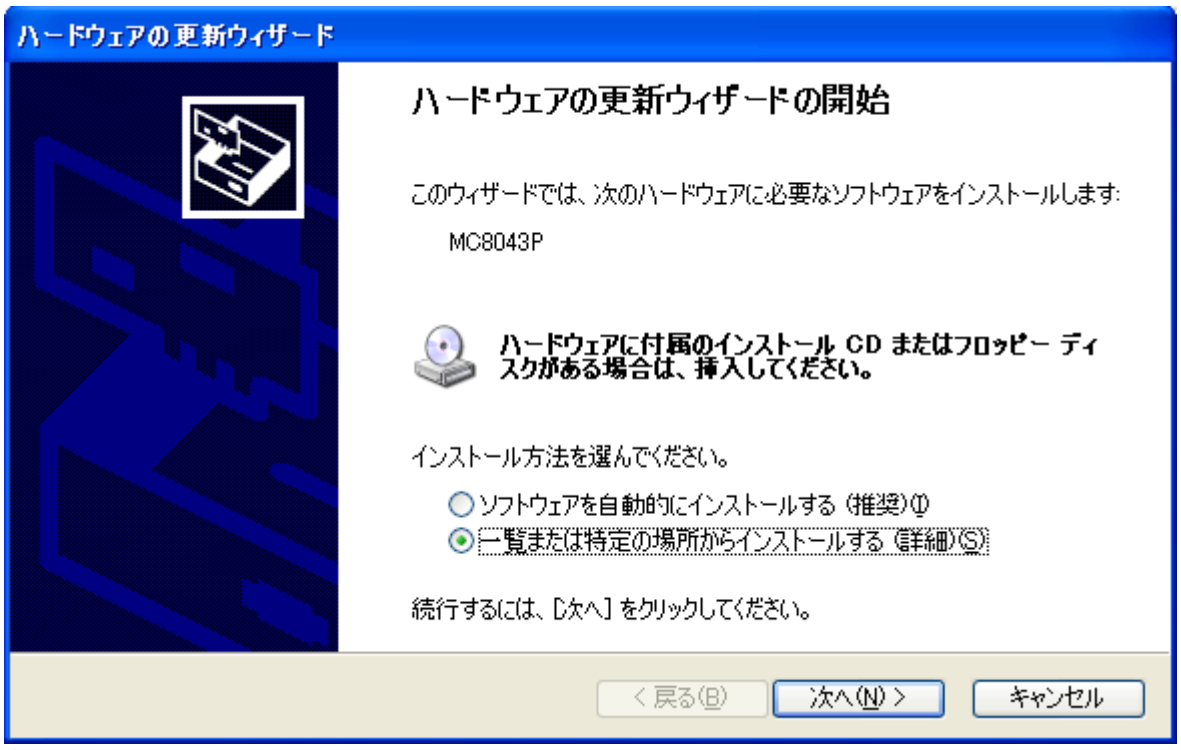




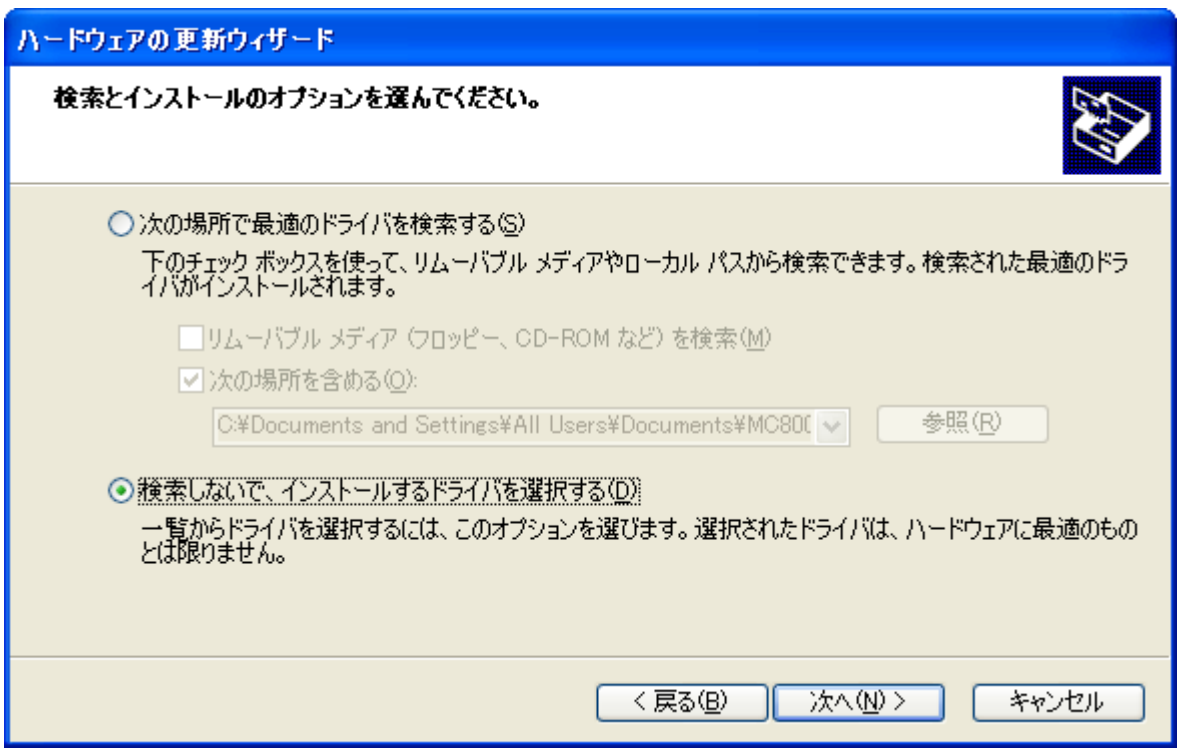
③Windows XP サービスパック２の場合は、下の画面が表示されますので、「いいえ、今回は接続しません」を選択し、
[次へ]をクリックします。
Windows XP サービスパック１の場合は、この画面は表示されませんので、次に進んで下さい。



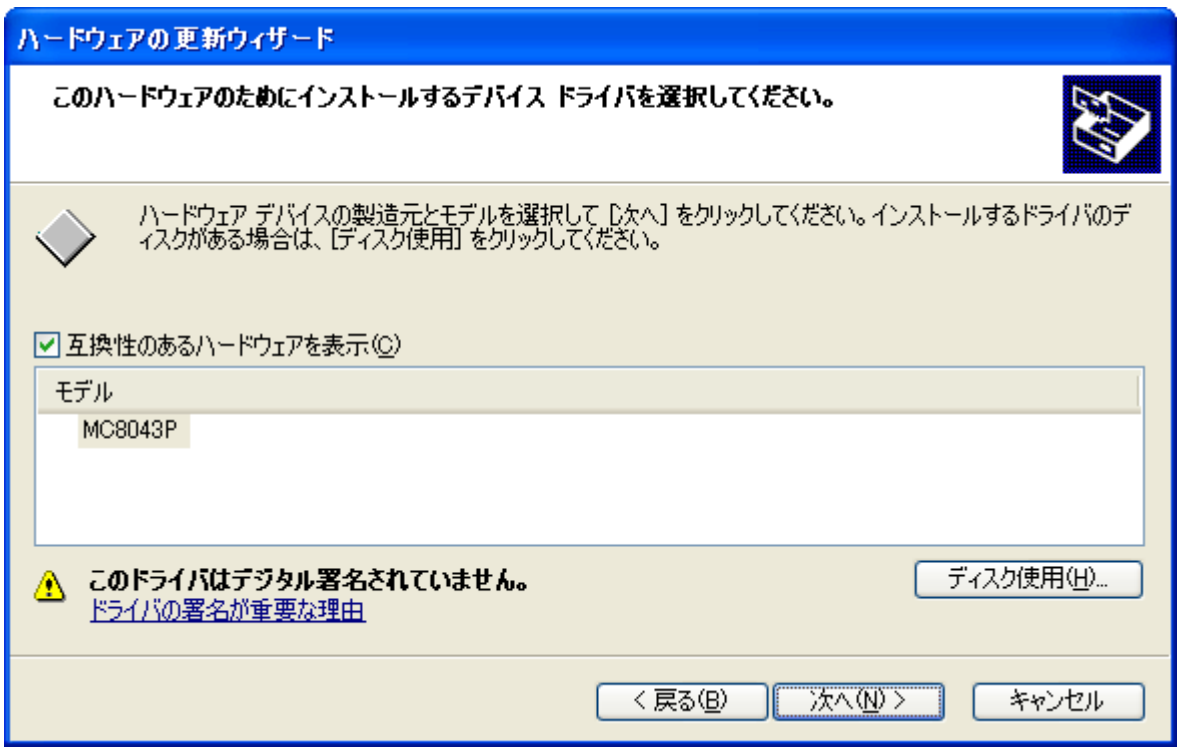
④下記画面で、「一覧または特定の場所からインストールする」を選択し、「次へ」ボタンをクリックします。



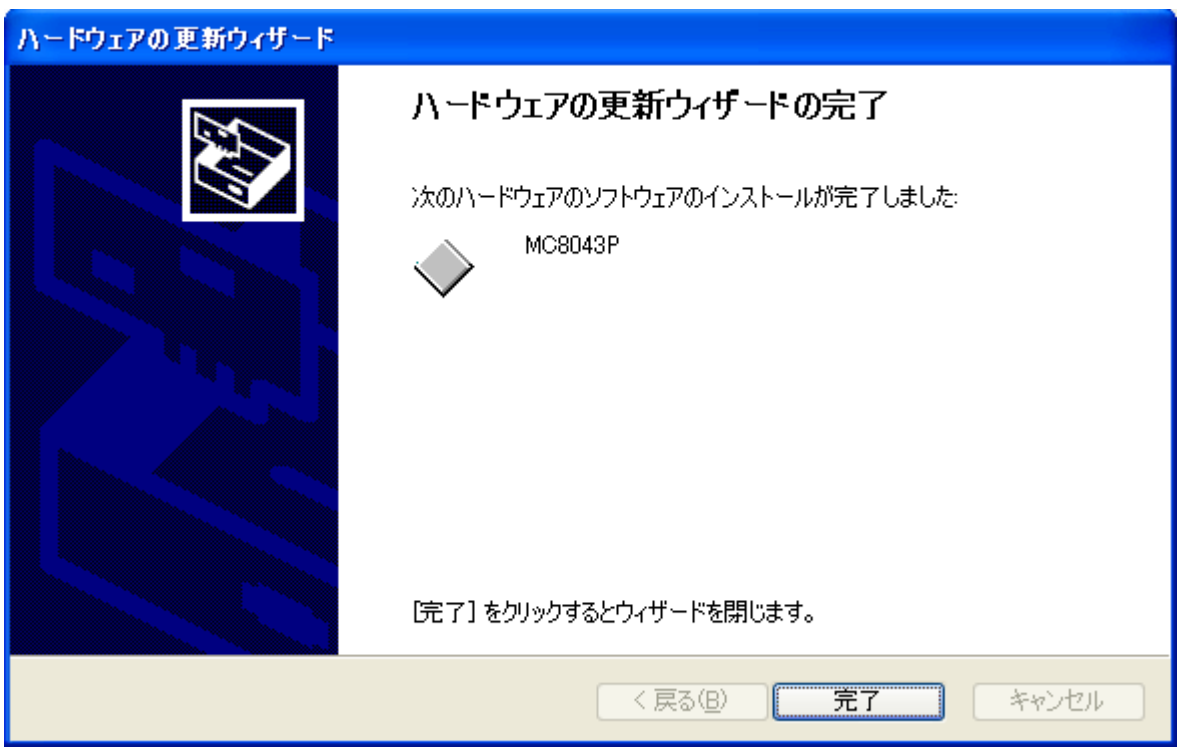
⑤下記画面で、「検索しないで、インストールするドライバを選択する」を選択し、[次へ]をクリックします。



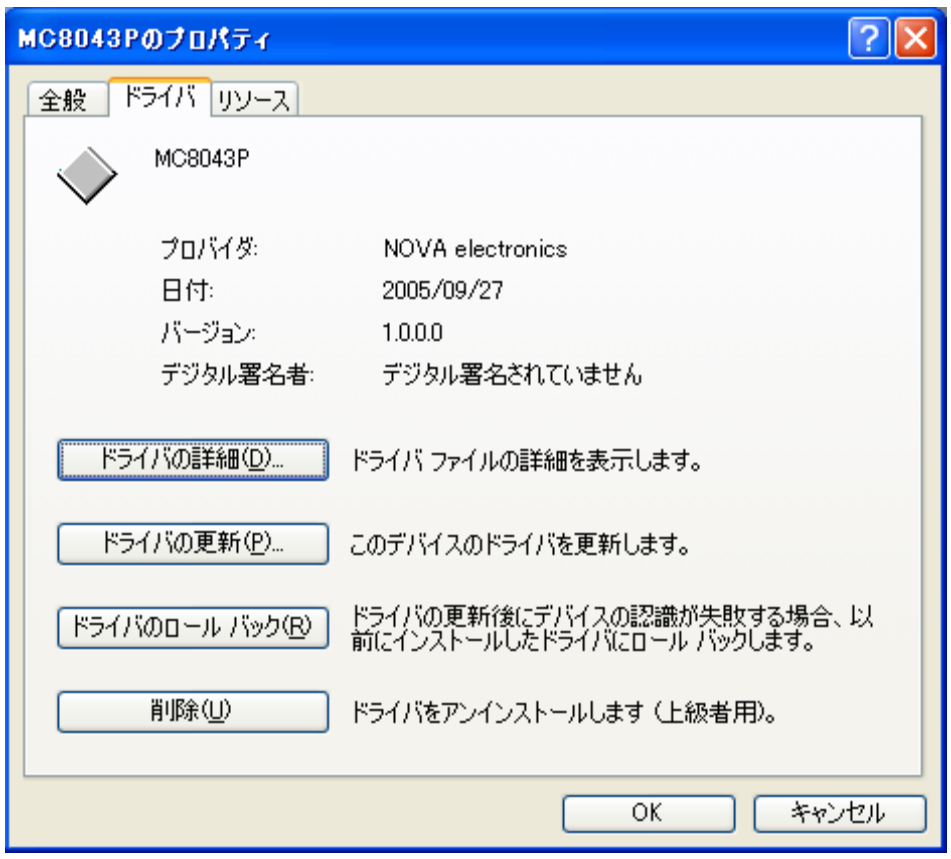
⑥下記画面で、「ディスク使用」をクリックし、参照ボタンから更新するドライバソフトのフォルダ（¥Driver）を選択後、
下記画面の[次へ]をクリックします。



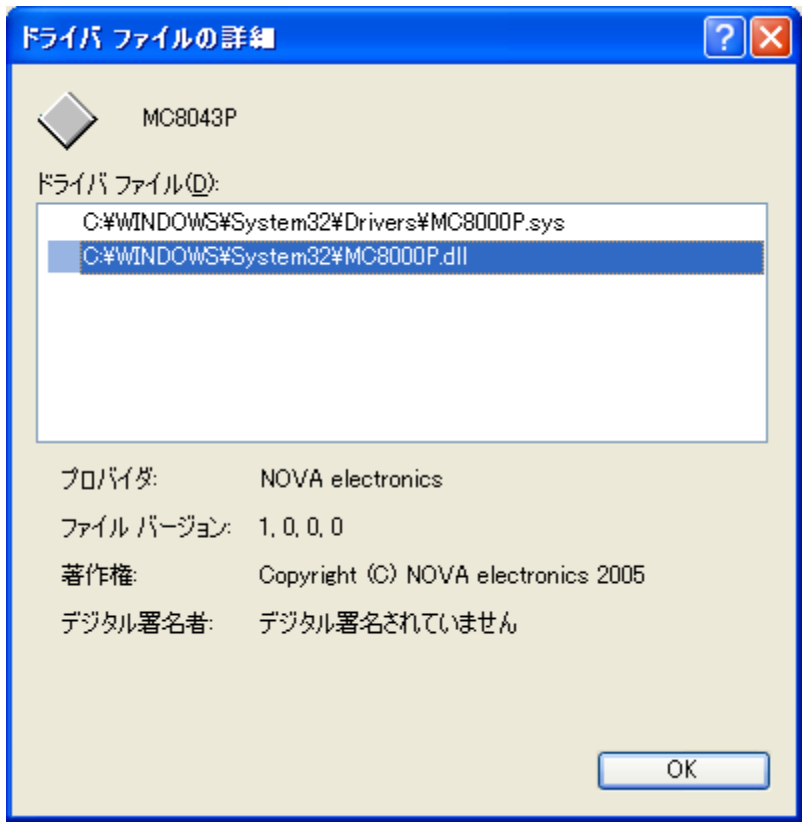
⑦正常に更新されると、下記画面が表示されます。



⑧下記画面（デバイスマネージャの更新したボードのプロパティ画面）で、更新したドライバのバージョンを確認します。
¥Driver¥Version.txt ファイルの「１．ドライババージョン」のバージョンが下記画面に表示されますので正しいか確認して下さい。 次に「ドライバの詳細」ボタンをクリックします。



⑨下記画面で、更新したドライバのファイルバージョンを確認します。
¥Driver¥Version.txt ファイルの「２．ドライバファイルバージョン」のバージョンが下記画面に表示されます。
Version.txt ファイルに記載している２つのファイルのバージョンが正しいか確認して下さい。



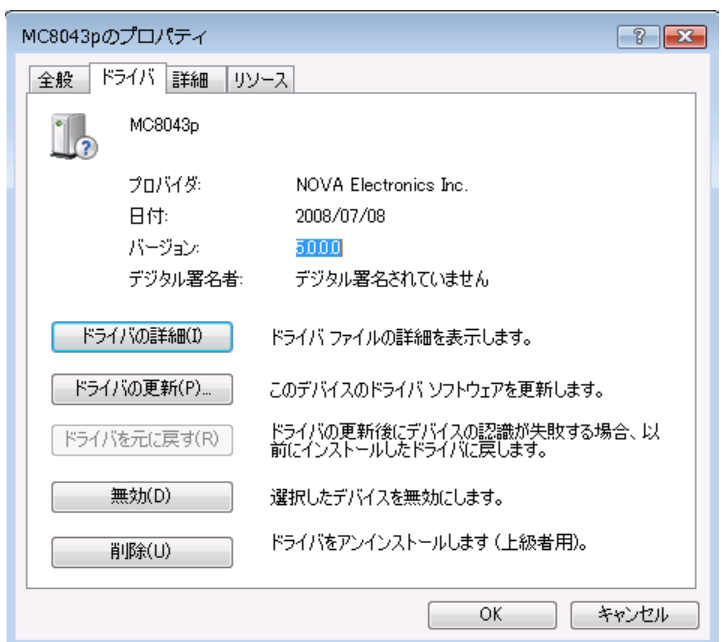
⑩ボードが複数ある場合は、「デバイスマネージャ」画面の「ＮＯＶＡ」の下に表示される全てのボードについて、①からの手順でドライバを更新して下さい。

⑪パソコンを再起動して下さい。これで更新作業は終了です。

2.6.4Windows Vista/7 (32bit/64bit)

起動中の他のアプリケーションは終了させてください。
デバイスドライバのインストールは必ずアドミニストレーター権限をもったユーザーログインで行ってください。アドミニストレーター権限以外でインストールをした場合正常にインストールされません。
下記の手順はWindows Vistaを例にしていますが、Windows 7でもデバイス更新の進め方は同じです。

① 2.1の方法でインストールするデバイスドライバを準備して下さい。
② Vista32ﾋﾞｯﾄは2.4.4の⑥から⑭を参照してインストールします。
Vista64ﾋﾞｯﾄは2.4.5の⑥から⑫を参照してインストールします。
③ 下記画面（デバイスマネージャの更新したボードのプロパティ画面）で、更新したドライバのバージョンを確認します。
¥Driver¥Version.txt ファイルの「１．ドライババージョン」のバージョンが下記画面に表示されますので正しいか確認して下さい。



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。
アプリケーション開発は、Microsoft Visual C++ (以下VC++)、Microsoft Visual Basic(以下VB)、あるいはMicrosoft Visual C# (以下C#)のいずれかを使用して行います。

3.1　動作環境

対応ＯＳ　　Windows 98(32bit) Windows 2000(32bit) Windows XP(32bit) Windows Vista(32bit,64bit) Windows 7(32bit,64bit)

対応言語

Microsoft Visual C++ 6.0
Microsoft Visual C++ .NET 2003
Microsoft Visual C++ 2005
Microsoft Visual C++ 2008
Microsoft Visual Basic 6.0
Microsoft Visual Basic .NET 2003
Microsoft Visual Basic 2005
Microsoft Visual Basic 2008
Microsoft Visual C#.NET 2003
Microsoft Visual C# 2005
Microsoft Visual C# 2008

注）アプリケーション開発を行う場合は、開発ツールのサポート状況などMSDNのサポートページを参考にソフトウェアを開発してください。サンプルプログラムをVisual Studio.NET2003、Visual Studio 2005、Visual Studio 2008で動作させる場合は、MSDNのサポートページを参考にサンプルプログラムの移行を行ってください。
64bitのデバイスドライバは64bitアプリにのみ対応しています。32bitアプリには対応しておりません。

3.2　ソフトウェア構成

3.2.1　ソフトウェアー覧

項目	フォルダ	ﾌｧｲﾙ名・ﾌｫﾙﾀﾞ名	説　　明
デバイスドライバ	Driver	MC8000P.SYS	デバイスドライバ本体
		MC8000P.DLL	ダイナミックリンクライブラリ VC++,VB,C# 共通
		MC8000P_8082.inf	インストールファイル(MC8082P,42P,22P及びMC8082Pe用)
		MC8000P_8080.inf	インストールファイル(MC8080P用)
		MC8000P_8043.inf	インストールファイル(MC8043P及びMC8043Pe用)
	Vista Driver	install MC8082P INF.exe	OSがVista以降のMC8082P,42P,22P及びMC8082Peデバイスドライバインストールプログラム
		install MC8080P INF.exe	OSがVista以降のMC8080Pデバイスドライバインストールプログラム
		install MC8043P INF.exe	OSがVista以降のMC8043P及びMC8043Peデバイスドライバインストールプログラム
		uninstall MC8082P INF.exe	OSがVista以降のMC8082P,42P,22P及びMC8082Peデバイスドライバアンインストールプログラム
		uninstall MC8080P INF.exe	OSがVista以降のMC8080Pデバイスドライバアンインストールプログラム
		uninstall MC8043P INF.exe	OSがVista以降のMC8043P及びMC8043Peデバイスドライバアンインストールプログラム
	Vista64 Driver	64bit install MC8082P INF.exe	OSがVista以降の64bitOS用MC8082P,42P,22P及びMC8082Peデバイスドライバインストールプログラム
		64bit install MC8080P INF.exe	OSがVista以降の64bitOS用MC8080Pデバイスドライバインストールプログラム
		64bit install MC8043P INF.exe	OSがVista以降の64bitOS用MC8043P及びMC8043Peデバイスドライバインストールプログラム
		64bit uninstall MC8082P INF.exe	OSがVista以降の64bitOS用MC8082P,42P,22P及びMC8082Peデバイスドライバアンインストールプログラム
		64bit uninstall MC8080P INF.exe	OSがVista以降の64bitOS用MC8080Pデバイスドライバアンインストールプログラム
		64bit uninstall MC8043P INF.exe	OSがVista以降の64bitOS用MC8043P及びMC8043Peデバイスドライバアンインストールプログラム

項目	フォルダ	ﾌｧｲﾙ名・ﾌｫﾙﾀﾞ名	説　　明
ライブラリ	Lib¥VB6	MC8000P_DLL.bas	MC8000P.DLLを使用する為のDeclare宣言ﾌｧｲﾙ VB6.0専用
	Lib¥VB.NET2003	MC8000P_DLL.vb	MC8000P.DLLを使用する為のDeclare宣言ﾌｧｲﾙ VB.NET2003専用
	Lib¥VC6 ¥Vista lib	MC8000P.LIB	MC8000P.DLLを使用するためのライブラリ OSがVista以降 VC++専用
		MC8000P_DLL.H	MC8000P.DLLを使用するためのヘッダ定義ファイル OSがVista以降 VC++専用
	Lib¥VC6 ¥Vista64 lib	MC8000P.LIB	MC8000P.DLLを使用するためのライブラリ OSがVista以降の64bitOSVC++専用
		MC8000P_DLL.H	MC8000P.DLLを使用するためのヘッダ定義ファイル OSがVista以降の64bitOS VC++専用
	Lib¥VC6	MC8000P.LIB	MC8000P.DLLを使用するためのライブラリ VC++専用
		MC8000P_DLL.H	MC8000P.DLLを使用するためのヘッダ定義ファイル VC++専用
	¥VB6		※MC8080P専用の旧関数を使用する場合のみ当ファイルを使用する
	Lib¥MC8080P_old ¥VB.NET2003	MC8080P_DLL.vb	MC8000P.DLLを使用する為のDeclare宣言ﾌｧｲﾙ VB.NET2003専用 ※MC8080P専用の旧関数を使用する場合のみ当ファイルを使用する
	Lib¥Csharp	Mc8000pWrap.dll	MC8000P.DLLを使用するためのクラスライブラリ C#専用
VBサンプルプログラム (VB6.0)	Sample¥ VB6	Sample A	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
		Sample C	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例 複数枚対応
		Sample D	全軸定量ドライブ、RR0,1,2,4,5読み出し例
		Sample F	補間関数を使用したＢＰ補間、連続補間例
		Sample G	自動原点出しのプログラム例
		Sample H	同期動作のプログラム例
		NormallyClose¥ Sample A	リミットセンサー論理 NormallyClose用プログラム リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
VBサンプルプログラム (VB.NET2003)	Sample¥ VB.NET2003	Sample A	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
		Sample C	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例 複数枚対応
		Sample D	全軸定量ドライブ、RR0,1,2,4,5読み出し例
		Sample F	補間関数を使用したＢＰ補間、連続補間例
		Sample G	自動原点出しのプログラム例
		Sample H	同期動作のプログラム例
		NormallyClose¥ Sample A	リミットセンサー論理 NormallyClose用プログラム リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
VC++サンプルプログラム (VC6.0)	Sample¥ VC6	Sample A	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
		Sample B	割り込みを使用したプログラム例
		Sample C	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作、 割り込みを使用したプログラム例、複数枚対応
		Sample D	全軸定量ドライブ、RR0,1,2,4,5読み出し、割込みプログラム例
		Sample E	割込みを用いた連続補間のプログラム例
		Sample F	補間関数を使用したＢＰ補間、連続補間例
		Sample G	自動原点出しのプログラム例
		Sample H	同期動作のプログラム例
		NormallyClose¥ Sample A	リミットセンサー論理 NormallyClose用プログラム リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
C#サンプルプログラム (C#.NET2003)	Sample¥ Csharp	Sample A	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例
		Sample B	割り込みを使用したプログラム例
		Sample C	リミット入力表示、論理位置カウンタ表示、定量ドライブ操作例 複数枚対応
		Sample D	全軸定量ドライブ、RR0,1,2,4,5を読み出し、割込みプログラム例
		Sample E	割込みを用いた連続補間のプログラム例
		Sample F	補間関数を使用したＢＰ補間、連続補間例
		Sample G	自動原点出しのプログラム例
		Sample H	同期動作のプログラム例

項目	フォルダ	ﾌｧｲﾙ名・ﾌｫﾙﾀﾞ名	説　　明
MCX304搭載ボード 評価ツール ※MC8082P,MC8080Pなど	Tool ¥MCX304 Board	MCX304-A.exe	MCX304搭載ボード評価ツール。画面からパラメータ、モード等を設定し、各コマンドを実行するアプリケーション。
		MCX304-B.exe	
		ParameterSample	パラメータサンプルファイル： MCX304-x.exeにてこのファイルをロードし、プロットを実行すると、各ファイル名称の動作をプロット画面上で確認できる
MCX314As搭載ボード 評価ツール ※MC8043Pなど	Tool ¥MCX314As Board	MCX314As-A.exe	MCX314As搭載ボード評価ツール。画面からパラメータ、モード等を設定し、各コマンドを実行するアプリケーション。
		MCX314As-B.exe	
		ParameterSample	パラメータサンプルファイル： MCX314As-x.exeにてこのファイルをロードし、プロットを実行すると、各ファイル名称の動作をプロット画面上で確認できる

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

3.2.2　ファイルの詳細

ソフトウェアのフォルダ構成とファイル内容を以下に示します。

注意：ＣＤ－ＲＯＭからハードディスクにコピーする場合、ファイルやフォルダが読み取り専用になる場合がありますので、必要であれば読み取り専用を解除してから使用して下さい。
MC8042P及びMC8022Pをご使用の場合、INFファイル及びVista以降のOSでのexeファイルについては、MC8082Pのものを使用します。つまり、INFファイルは、MC8000P_8082P.infです。exeファイルは、install MC8082P INF.exeというようになります。
MC8043Peをご使用の場合、INFファイル及びVista以降のOSでのexeファイルについては、MC8043Pのものを使用します。つまり、INFファイルは、MC8000P_8043P.infです。exeファイルは、install MC8043P INF.exeというようになります。
MC8082Peをご使用の場合、INFファイル及びVista以降のOSでのexeファイルについては、MC8082Pのものを使用します。つまり、INFファイルは、MC8000P_8082P.infです。exeファイルは、install MC8082P INF.exeというようになります。

```
¥
+---Driver
|    +---Driver                          OSがXP以前の32bitOS用
|    |   +---MC8000P.sys                 デバイスドライバ本体
|    |   +---MC8000P.dll                 デバイスドライバを使用するためのダイナミックリンクライブラリ
|    |   +---MC8000P_8082.inf            デバイスドライバのインストール用プログラム(MC8082P,42P,22P及びMC8082Pe用)
|    |   +---MC8000P_8080.inf            デバイスドライバのインストール用プログラム(MC8080P用)
|    |   +---MC8000P_8043.inf            デバイスドライバのインストール用プログラム(MC8043P及びMC8043Pe用)
|    |   +---Version.txt                 デバイスドライバのバージョン説明ファイル
|    |
|    +---Vista Driver                    OSがVista以降の32bitOS用
|    |   +---install MC8082P INF.exe     デバイスドライバのインストール用プログラム(MC8082P,42P,22P及びMC8082Pe用)
|    |   +---install MC8080P INF.exe     デバイスドライバのインストール用プログラム(MC8080P用)
|    |   +---install MC8043P INF.exe     デバイスドライバのインストール用プログラム(MC8043P用及びMC8043Pe用)
|    |   +---uninstall MC8082P INF.exe   デバイスドライバのアンインストール用プログラム
|    |                                     (MC8082P,42P,22P及びMC8082Pe用)
|    |   +---uninstall MC8080P INF.exe   デバイスドライバのアンインストール用プログラム(MC8080P用)
|    |   +---uninstall MC8043P INF.exe   デバイスドライバのアンインストール用プログラム(MC8043P用及びMC8043Pe用)
|    |   +---Version.txt                 デバイスドライバのバージョン説明ファイル
|    |
|    +---Vista64 Driver                          OSがVista以降の64bitOS用
|        +---64bit install MC8082P INF.exe       デバイスドライバのインストール用プログラム
|                                                  (MC8082P,42P,22P用及びMC8082Pe用)
|        +---64bit install MC8080P INF.exe       デバイスドライバのインストール用プログラム(MC8080P用)
|        +---64bit install MC8043P INF.exe       デバイスドライバのインストール用プログラム(MC8043P及びMC8043Pe用)
|        +---64bit uninstall MC8082P INF.exe     デバイスドライバのアンインストール用プログラム
|                                                  (MC8082P,42P,22P用及びMC8082Pe用)
|        +---64bit uninstall MC8080P INF.exe     デバイスドライバのアンインストール用プログラム(MC8080P用)
|        +---64bit uninstall MC8043P INF.exe     デバイスドライバのアンインストール用プログラム
|                                                  (MC8043P及びMC8043Pe用)
|        +---Version.txt                         デバイスドライバのバージョン説明ファイル
|
+---LIB
|    +---VB6
|    |   +---MC8000P_DLL.bas             VB6.0用 MC8000P.DLL Declare宣言、各定義ファイル
|    +---VB.NET2003
|    |   +---MC8000P_DLL.vb              VB.NET2003用 MC8000P.DLL Declare宣言、各定義ファイル
|    +---VC6
|    |   +---Vista lib                   OSがVista以降の32bitOS用
|    |   |   +---MC8000P.lib             VC6.0用 MC8000P.DLLのライブラリファイル（Vista用）
|    |   |   +---MC8000P_DLL.h           VC6.0用 MC8000P.DLLのヘッダファイル（関数宣言、各定義）（Vista用）
|    |   +---Vista64 lib                 OSがVista以降の64bitOS用
|    |   |   +---MC8000P.lib             VC6.0用 MC8000P.DLLのライブラリファイル
|    |   |   +---MC8000P_DLL.h           VC6.0用 MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|    |   +---MC8000P.lib                 VC6.0用 MC8000P.DLLのライブラリファイル(OSがXP以前の32bitOS用)
|    |   +---MC8000P_DLL.h               VC6.0用 MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|    |                                     (OSがXP以前の32bitOS用)
|    |
|    +---MC8080P_old
|    |   +---VB6
|    |   |   +---MC8000P_DLL.bas         VB6.0用 MC8000P.DLL Declare宣言、各定義ファイル(MC8080P専用の旧関数用)
|    |   +---VB.NET2003
|    |       +---MC8000P_DLL.vb          VB.NET2003用 MC8000P.DLL Declare宣言、各定義ﾌｧｲﾙ(MC8080P専用の旧関数用)
|    +---Csharp
|        +---Mc8000pWrap.dll             C#.NET2003用 MC8000P Ｃ＃クラスライブラリファイル
|
```

```
|
+---Tool
|    +---MCX304 Board
|    |   +---MCX304-A.exe                MCX304搭載ボード評価ツール(IC-A)
|    |   +---MCX304-B.exe                MCX304搭載ボード評価ツール(IC-B)
|    |   +---ReadMe.txt                  操作説明書
|    |   +---ParameterSample             パラメータサンプルファイル
|    |       +---対称Ｓ字加減速-1.msd    対称Ｓ字加減速パラメータファイル１
|    |       +---対称Ｓ字加減速-2.msd    対称Ｓ字加減速パラメータファイル２
|    |       +---対称台形加減速-1.msd    対称台形加減速パラメータファイル１
|    |       +---対称台形加減速-2.msd    対称台形加減速パラメータファイル２
|    |       +---非対称台形加減速-1.msd  非対称台形加減速パラメータファイル１
|    |       +---非対称台形加減速-2.msd  非対称台形加減速パラメータファイル２
|    |       +---readme.txt              説明書
|    |
|    +---MCX314As Board
|        +---MCX314As-A.exe              MCX314As搭載ボード評価ツール(IC-A)
|        +---MCX314As-B.exe              MCX314As搭載ボード評価ツール(IC-B)
|        +---ReadMe.txt                  操作説明書
|        +---ParameterSample             パラメータサンプルファイル
|            +---対称Ｓ字加減速-1.msd    対称Ｓ字加減速パラメータファイル１
|            +---対称Ｓ字加減速-2.msd    対称Ｓ字加減速パラメータファイル２
|            +---対称台形加減速-1.msd    対称台形加減速パラメータファイル１
|            +---対称台形加減速-2.msd    対称台形加減速パラメータファイル２
|            +---非対称Ｓ字加減速-1.msd  非対称Ｓ字加減速パラメータファイル１
|            +---非対称Ｓ字加減速-2.msd  非対称Ｓ字加減速パラメータファイル２
|            +---非対称台形加減速-1.msd  非対称台形加減速パラメータファイル１
|            +---非対称台形加減速-2.msd  非対称台形加減速パラメータファイル２
|            +---readme.txt              説明書
|
+---Sample
    +---VB6
    |   +---NormallyClose
    |   |   +---Sample A
    |   |      +---Form1.frm             サンプルプログラムＡ (Normally Close)
    |   |      +---MC8000P_DLL.bas       VB6用MC8000P.DLL Declare宣言、各定義ファイル
    |   |      +---Module1.bas           ボード制御関数サンプル
    |   |      +---VBSample.vbp          VBサンプルプログラム用プロジェクトファイル (VB6.0)
    |   |      +---exe
    |   |          +---VBSampleA.exe     実行ファイル
    |   |
    |   +---Sample A
    |   |   +---Form1.frm                サンプルプログラムＡ
    |   |   +---MC8000P_DLL.bas          VB6用MC8000P.DLL Declare宣言、各定義ファイル
    |   |   +---Module1.bas              ボード制御関数サンプル
    |   |   +---VBSample.vbp             VBサンプルプログラム用プロジェクトファイル (VB6.0)
    |   |   +---exe
    |   |       +---VBSampleA.exe        実行ファイル
    |   |
    |   +---Sample C
    |   |   +---Form1.frm                サンプルプログラムＣ
    |   |   +---MC8000P_DLL.bas          VB6用MC8000P.DLL Declare宣言、各定義ファイル
    |   |   +---Module1.bas              ボード制御関数サンプル
    |   |   +---VBSample.vbp             VBサンプルプログラム用プロジェクトファイル (VB6.0)
    |   |   +---exe
    |   |       +---VBSampleC.exe        実行ファイル
    |   |
    |   +---Sample D
    |   |   +---Form1.frm                サンプルプログラムＤ
    |   |   +---MC8000P_DLL.bas          VB6用MC8000P.DLL Declare宣言、各定義ファイル
    |   |   +---Module1.bas              ボード制御関数サンプル
    |   |   +---VBSample.vbp             VBサンプルプログラム用プロジェクトファイル (VB6.0)
    |   |   +---exe
    |   |       +---VBSampleD.exe        実行ファイル
    |   |
    |   |
    |   |
    |   |
    |   |
    |   |
```

```
|   |
|   +---Sample F
|   |   +---Form1.frm               サンプルプログラムＦ
|   |   +---MC8000P_DLL.bas         VB6用MC8000P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.bas             ボード制御関数サンプル
|   |   +---VBSample.vbp            VBサンプルプログラム用プロジェクトファイル（VB6.0）
|   |   +---exe
|   |       +---VBSampleF.exe       実行ファイル
|   |
|   +---Sample G
|   |   +---Form1.frm               サンプルプログラムＧ
|   |   +---MC8000P_DLL.bas         VB6用MC8000P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.bas             ボード制御関数サンプル
|   |   +---VBSample.vbp            VBサンプルプログラム用プロジェクトファイル（VB6.0）
|   |   +---exe
|   |       +---VBSampleG.exe 実行ファイル
|   |
|   +---Sample H
|       +---Form1.frm               サンプルプログラムＨ
|       +---MC8000P_DLL.bas         VB6用MC8000P.DLL Declare宣言、各定義ファイル
|       +---Module1.bas             ボード制御関数サンプル
|       +---VBSample.vbp            VBサンプルプログラム用プロジェクトファイル（VB6.0）
|       +---exe
|           +---VBSampleH.exe 実行ファイル
|
+---VB.NET2003
|   +---NormallyClose
|   |   +---Sample A
|   |      +---Form1.vb             サンプルプログラムＡ（Normally Close）
|   |      +---MC8000P_DLL.vb VB.NET2003用 MC8000P.DLL Declare宣言、各定義ファイル
|   |      +---Module1.vb           ボード制御関数サンプル
|   |      +---VBSample.sln         VBサンプルプログラム用ソリューションファイル（VB.NET2003）
|   |      +---exe
|   |          +---VBSampleA.exe    実行ファイル
|   |
|   +---Sample A
|   |   +---Form1.vb                サンプルプログラムＡ
|   |   +---MC8000P_DLL.vb          VB.NET2003用 MC8000P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.vb              ボード制御関数サンプル
|   |   +---VBSample.sln            VBサンプルプログラム用ソリューションファイル（VB.NET2003）
|   |   +---exe
|   |       +---VBSampleA.exe       実行ファイル
|   |
|   +---Sample C
|   |   +---FormC.vb                サンプルプログラムＣ
|   |   +---MC8000P_DLL.vb          VB.NET2003用 MC8000P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.vb              ボード制御関数サンプル
|   |   +---VBSample.sln            VBサンプルプログラム用ソリューションファイル（VB.NET2003）
|   |   +---exe
|   |       +---VBSampleC.exe       実行ファイル
|   |
|   +---Sample D
|   |   +---Form1.vb                サンプルプログラムＤ
|   |   +---MC8000P_DLL.vb          VB.NET2003用 MC8000P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.vb              ボード制御関数サンプル
|   |   +---VBSample.sln            VBサンプルプログラム用ソリューションファイル（VB.NET2003）
|   |   +---exe
|   |       +---VBSampleD.exe       実行ファイル
|   |
|   +---Sample F
|   |   +---Form1.vb                サンプルプログラムＦ
|   |   +---MC8000P_DLL.vb          VB.NET2003用 MC8000P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.vb              ボード制御関数サンプル
|   |   +---VBSample.sln            VBサンプルプログラム用ソリューションファイル（VB.NET2003）
|   |   +---exe
|   |       +---VBSampleF.exe       実行ファイル
|   |
|   |
|   |
|   |
```

```
|   +---Sample G
|   |   +---Form1.vb                サンプルプログラムＧ
|   |   +---MC8000P_DLL.vb          VB.NET2003用 MC8000P.DLL Declare宣言、各定義ファイル
|   |   +---Module1.vb              ボード制御関数サンプル
|   |   +---VBSample.sln            VBサンプルプログラム用ソリューションファイル（VB.NET2003）
|   |   +---exe
|   |       +---VBSampleG.exe       実行ファイル
|   |
|   +---Sample H
|       +---Form1.vb                サンプルプログラムＨ
|       +---MC8000P_DLL.vb          VB.NET2003用 MC8000P.DLL Declare宣言、各定義ファイル
|       +---Module1.vb              ボード制御関数サンプル
|       +---VBSample.sln            VBサンプルプログラム用ソリューションファイル（VB.NET2003）
|       +---exe
|           +---VBSampleH.exe       実行ファイル
|
+---VC6
|   +---NormallyClose
|   |   +---Sample A                サンプルプログラムＡ（Normally Close）
|   |       +---VCSample.cpp        アプリケーションクラスメンバ関数
|   |       +---VCSampleDlg.cpp     ダイアログクラスメンバ関数
|   |       +---VCSample.h          アプリケーションクラス宣言
|   |       +---VCSampleDlg.h       ダイアログクラス宣言
|   |       +---MC8000P.cpp         ボード制御関数サンプル
|   |       +---MC8000P.H           ボード制御関数宣言
|   |       +---MC8000P.LIB         MC8000P.DLLのライブラリファイル
|   |       +---MC8000P_DLL.H       MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|   |       +---VCSample.dsw        VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |       +---exe
|   |           +---VCSampleA.exe   実行ファイル
|   |
|   +---Sample A                    サンプルプログラムＡ
|   |   +---VCSample.cpp            アプリケーションクラスメンバ関数
|   |   +---VCSampleDlg.cpp         ダイアログクラスメンバ関数
|   |   +---VCSample.h              アプリケーションクラス宣言
|   |   +---VCSampleDlg.h           ダイアログクラス宣言
|   |   +---MC8000P.cpp             ボード制御関数サンプル
|   |   +---MC8000P.H               ボード制御関数宣言
|   |   +---MC8000P.LIB             MC8000P.DLLのライブラリファイル
|   |   +---MC8000P_DLL.H           MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---VCSample.dsw            VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---exe
|   |       +---VCSampleA.exe       実行ファイル
|   |
|   +---Sample B                    サンプルプログラムＢ
|   |   +---VCSample.cpp            アプリケーションクラスメンバ関数
|   |   +---VCSampleDlg.cpp         ダイアログクラスメンバ関数
|   |   +---VCSample.h              アプリケーションクラス宣言
|   |   +---VCSampleDlg.h           ダイアログクラス宣言
|   |   +---MC8000P.cpp             ボード制御関数サンプル
|   |   +---MC8000P.H               ボード制御関数宣言
|   |   +---MC8000P.LIB             MC8000P.DLLのライブラリファイル
|   |   +---MC8000P_DLL.H           MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---VCSample.dsw            VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---exe
|   |       +---VCSampleB.exe       実行ファイル
|   |
|   +---Sample C                    サンプルプログラムＣ
|   |   +---VCSample.cpp            アプリケーションクラスメンバ関数
|   |   +---VCSampleDlg.cpp         ダイアログクラスメンバ関数
|   |   +---VCSample.h              アプリケーションクラス宣言
|   |   +---VCSampleDlg.h           ダイアログクラス宣言
|   |   +---MC8000P.cpp             ボード制御関数サンプル
|   |   +---MC8000P.H               ボード制御関数宣言
|   |   +---MC8000P.LIB             MC8000P.DLLのライブラリファイル
|   |   +---MC8000P_DLL.H           MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---VCSample.dsw            VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---exe
|   |       +---VCSampleC.exe       実行ファイル
|   |
```

```
|   |
|   +---Sample D                    サンプルプログラムＤ
|   |   +---MC_SAMPLE.cpp           アプリケーションクラスメンバ関数
|   |   +---MC_SAMPLEDlg.cpp        ダイアログクラスメンバ関数
|   |   +---MC_SAMPLE.H             アプリケーションクラス宣言
|   |   +---MC_SAMPLEDlg.H          ダイアログクラス宣言
|   |   +---MC8000P.cpp             ボード制御関数サンプル
|   |   +---MC8000P.H               ボード制御関数宣言
|   |   +---MC8000P.LIB             MC8000P.DLLのライブラリファイル
|   |   +---MC8000P_DLL.H           MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---MC_SAMPLE.dsw           VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---exe
|   |       +---VCSampleD.exe       実行ファイル
|   |
|   +---Sample E                    サンプルプログラムＥ
|   |   +---MC_Sample2.cpp          アプリケーションクラスメンバ関数
|   |   +---MC_Sample2Dlg.cpp       ダイアログクラスメンバ関数
|   |   +---MC_Sample2.H            アプリケーションクラス宣言
|   |   +---MC_Sample2Dlg.H         ダイアログクラス宣言
|   |   +---MC8000P.cpp             ボード制御関数サンプル
|   |   +---MC8000P.H               ボード制御関数宣言
|   |   +---MC8000P.LIB             MC8000P.DLLのライブラリファイル
|   |   +---MC8000P_DLL.H           MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---MC_Sample2.dsw          VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---Sample.bmp              このアプリケーションで実行する連続補間の軌跡図
|   |   +---exe
|   |       +---VCSampleE.exe       実行ファイル
|   |
|   +---Sample F                    サンプルプログラムＦ
|   |   +---MC_SAMPLE.cpp           アプリケーションクラスメンバ関数
|   |   +---MC_SAMPLEDlg.cpp        ダイアログクラスメンバ関数
|   |   +---MC_SAMPLE.H             アプリケーションクラス宣言
|   |   +---MC_SAMPLEDlg.H          ダイアログクラス宣言
|   |   +---MC8000P.cpp             ボード制御関数サンプル
|   |   +---MC8000P.H               ボード制御関数宣言
|   |   +---MC8000P.LIB             MC8000P.DLLのライブラリファイル
|   |   +---MC8000P_DLL.H           MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---MC_SAMPLE.dsw           VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---exe
|   |       +---VCSampleF.exe       実行ファイル
|   |
|   +---Sample G                    サンプルプログラムＧ
|   |   +---MC_SAMPLE.cpp           アプリケーションクラスメンバ関数
|   |   +---MC_SAMPLEDlg.cpp        ダイアログクラスメンバ関数
|   |   +---MC_SAMPLE.H             アプリケーションクラス宣言
|   |   +---MC_SAMPLEDlg.H          ダイアログクラス宣言
|   |   +---MC8000P.cpp             ボード制御関数サンプル
|   |   +---MC8000P.H               ボード制御関数宣言
|   |   +---MC8000P.LIB             MC8000P.DLLのライブラリファイル
|   |   +---MC8000P_DLL.H           MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|   |   +---MC_SAMPLE.dsw           VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|   |   +---exe
|   |       +---VCSampleG.exe       実行ファイル
|   |
|   +---Sample H                    サンプルプログラムＨ
|       +---MC_SAMPLE.cpp           アプリケーションクラスメンバ関数
|       +---MC_SAMPLEDlg.cpp        ダイアログクラスメンバ関数
|       +---MC_SAMPLE.H             アプリケーションクラス宣言
|       +---MC_SAMPLEDlg.H          ダイアログクラス宣言
|       +---MC8000P.cpp             ボード制御関数サンプル
|       +---MC8000P.H               ボード制御関数宣言
|       +---MC8000P.LIB             MC8000P.DLLのライブラリファイル
|       +---MC8000P_DLL.H           MC8000P.DLLのヘッダファイル（関数宣言、各定義）
|       +---MC_SAMPLE.dsw           VCサンプルプログラム用プロジェクトワークスペース（VC6.0のみ）
|       +---exe
|           +---VCSampleH.exe       実行ファイル
|
|
|
|
```

```
|
+---Csharp
    +---Sample A                サンプルプログラムＡ
    |   +---Form1.cs            サンプルプログラムＡ Ｃ＃ソースファイル
    |   +---SampleA.csproj      サンプルプログラムＡプロジェクトワークスペース（C#のみ）
    |   +---exe
    |       +---SampleA.exe     サンプルＡ実行ファイル
    |
    +---Sample B                サンプルプログラムＢ
    |   +---Form1.cs            サンプルプログラムＢ Ｃ＃ソースファイル
    |   +---SampleB.csproj      サンプルプログラムＢプロジェクトワークスペース（C#のみ）
    |   +---exe
    |       +---SampleB.exe     サンプルＢ実行ファイル
    |
    +---Sample C                サンプルプログラムＣ
    |   +---Form1.cs            サンプルプログラムＣ Ｃ＃ソースファイル
    |   +---SampleC.csproj      サンプルプログラムＣプロジェクトワークスペース（C#のみ）
    |   +---exe
    |       +---SampleC.exe     サンプルＣ実行ファイル
    |
    +---Sample D                サンプルプログラムＤ
    |   +---Form1.cs            サンプルプログラムＤ Ｃ＃ソースファイル
    |   +---SampleD.csproj      サンプルプログラムＤプロジェクトワークスペース（C#のみ）
    |   +---exe
    |       +---SampleD.exe     サンプルＤ実行ファイル
    |
    +---Sample E                サンプルプログラムＥ
    |   +---Form1.cs            サンプルプログラムＥ Ｃ＃ソースファイル
    |   +---SampleE.csproj      サンプルプログラムＥプロジェクトワークスペース（C#のみ）
    |   +---exe
    |       +---SampleE.exe     サンプルＥ実行ファイル
    |
    +---Sample F                サンプルプログラムＦ
    |   +---Form1.cs            サンプルプログラムＦ Ｃ＃ソースファイル
    |   +---SampleF.csproj      サンプルプログラムＦプロジェクトワークスペース（C#のみ）
    |   +---exe
    |       +---SampleF.exe     サンプルＦ実行ファイル
    |
+---Sample G                    サンプルプログラムＧ
|   +---Form1.cs                サンプルプログラムＧ Ｃ＃ソースファイル
|   +---SampleG.csproj          サンプルプログラムＧプロジェクトワークスペース（C#のみ）
|   +---exe
|       +---SampleG.exe         サンプルＧ実行ファイル
|
+---Sample H                    サンプルプログラムＨ
    +---Form1.cs                サンプルプログラムＨ Ｃ＃ソースファイル
    +---SampleH.csproj          サンプルプログラムＨプロジェクトワークスペース（C#のみ）
    +---exe
        +---SampleH.exe         サンプルＨ実行ファイル
```

3.3　開発手順

3.3.1　ＶＣ＋＋の場合（ VC++6.0, VC++.NET2003, VC++ 2005, VC++ 2008 ）

アプリケーションはMC8000P.libとMC8000P_DLL.hファイルを使用します。この２ファイルは VC++ 6.0 以降対応です。

① ¥Lib¥VC6フォルダに入っている２つのファイルMC8000P.libとMC8000P_DLL.hを開発するアプリケーションのフォルダにコピーしてください。

② VC++の総合開発環境にてMC8000P_DLL.h をご使用のプロジェクトに追加登録してください。
また、API 関数を使用するソースファイルにMC8000P_DLL.h をincludeして下さい。

③ VC++6.0 の場合は、[プロジェクト]－[設定]で「リンク」タブを選択し「オブジェクト／ライブラリモジュール」にMC8000P.libを追加して下さい。（「図3.3-1 VC++6.0 プロジェクトの設定」を参照。）

VC++.NET2003、VC++ 2005、VC++ 2008の場合は、[プロジェクト]－[プロパティ]画面で[リンカ]－[入力]を選択し「追加の依存ファイル」にMC8000P.libを追加して下さい。
（「図3.3-2 VC++.NET2003 プロジェクトのプロパティ」、「図3.3-3 VC++ 2005 プロジェクトのプロパティ」「図3.3-4 VC++ 2008 プロジェクトのプロパティ」を参照。）

④ 3.4 ＡＰＩの関数を使用してプログラミングを行って下さい。

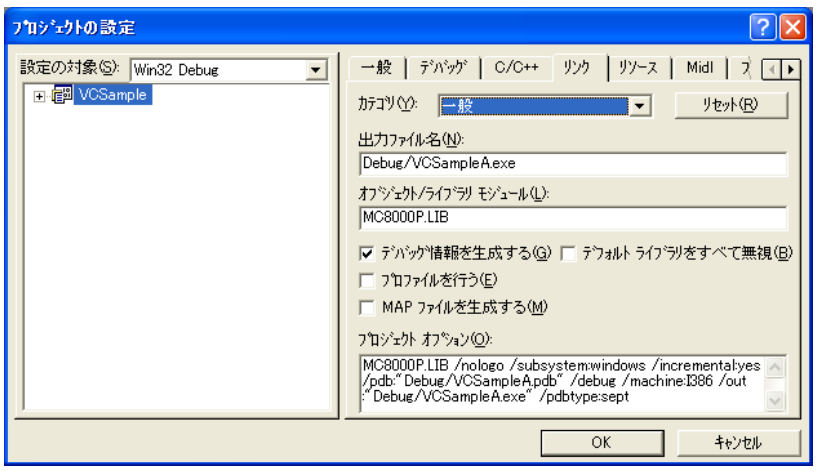


図 3.3-1 VC++6.0 プロジェクトの設定

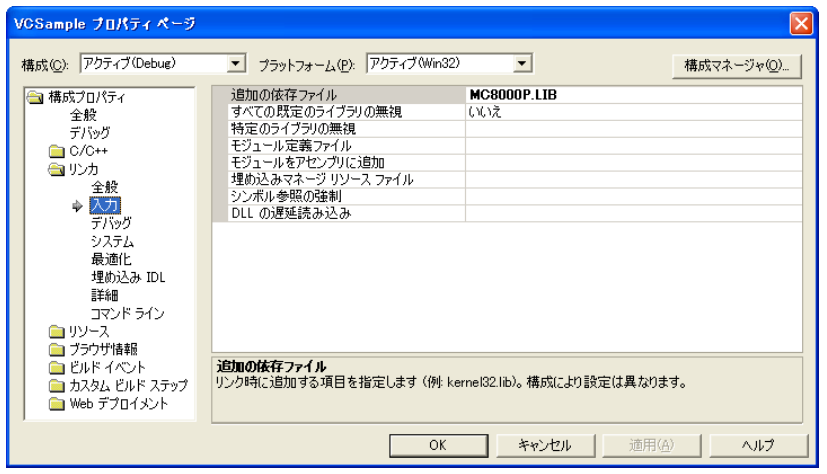


図 3.3-2 VC++.NET2003 プロジェクトのプロパティ

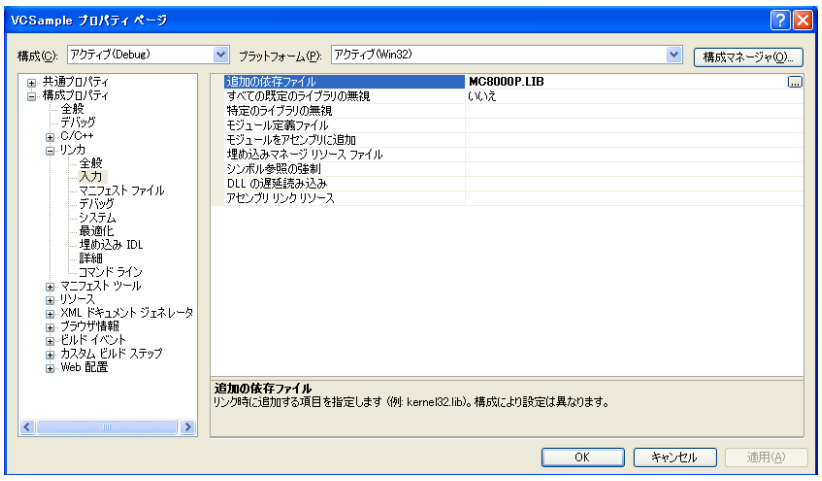


図3.3-3 VC++ 2005 プロジェクトのプロパティ

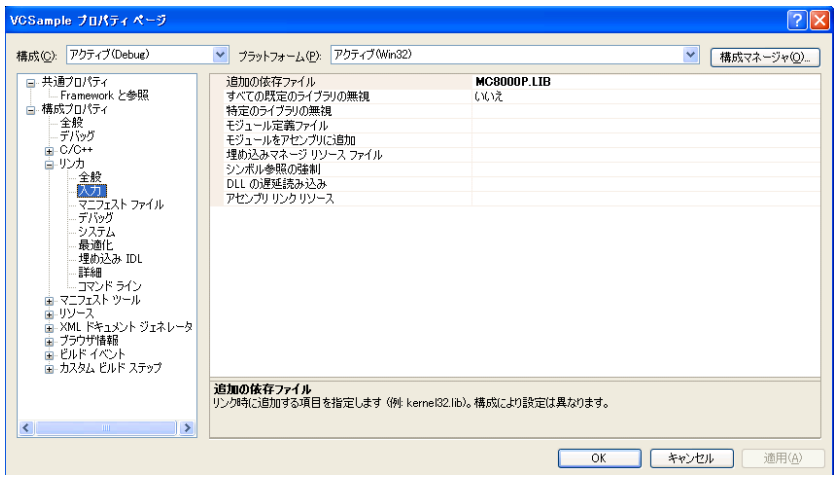


図3.3-4 VC++ 2008 プロジェクトのプロパティ

3.3.2　ＶＢ６.０の場合

① ¥Lib¥VB6 フォルダに入っているMC8000P_DLL.BASファイルを開発するアプリケーションのプロジェクトに標準モジュールとして
　追加してください。

② 3.4 ＡＰＩの関数を使用してプログラミングを行って下さい。

注意：VB では、本ボードに関する割り込みを使用することはできません。

3.3.3　ＶＢ.ＮＥＴ２００３、ＶＢ ２００５、ＶＢ ２００８の場合

① ¥Lib¥VB.NET2003 フォルダに入っているMC8000P_DLL.vbファイルを開発するアプリケーションのプロジェクトに追加してください。

② 3.4 ＡＰＩの関数を使用してプログラミングを行って下さい。

注意：VB では、本ボードに関する割り込みを使用することはできません。

3.3.4　Ｃ＃の場合

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

①　¥LIB¥Csharpフォルダに入っているファイルMc8000pWrap.dllを開発するアプリケーションのフォルダにコピーしてください。

②　C#.NET2003の場合は[プロジェクト]－[参照の追加]で、Mc8000pWrap.dllへの参照を追加してください。
（「図3.3-5 C#.NET2003 参照の追加」を参照。）

C# 2005、C# 2008の場合は、[プロジェクト]－[参照の追加]で「参照」タブを選択しMc8000pWrap.dllへの参照を追加して下さい。
（「図3.3-6 C# 2005、C# 2008 参照の追加」を参照。）

③　アプリケーションのソースファイルにusingでネームスペース　Mc8000pWrap　を追加してください。

④　3.4 ＡＰＩ（ MC8000Pドライバ関数 ）の関数を使用してプログラミングを行って下さい。

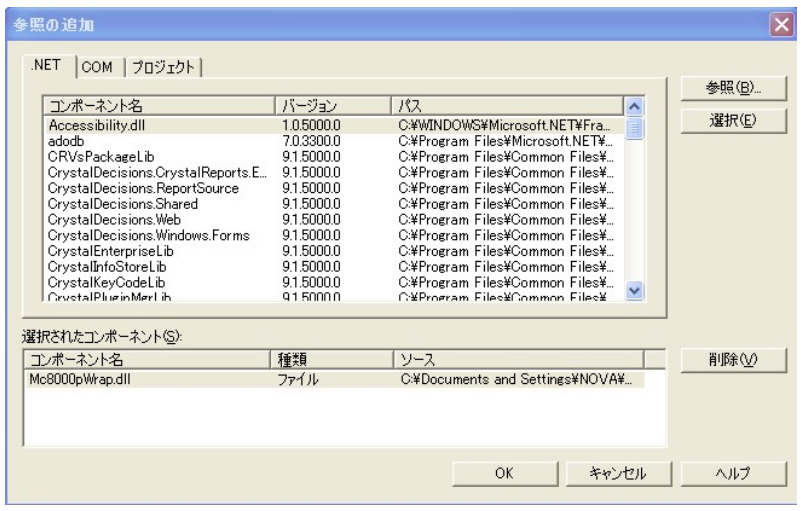


図 3.3-5 C#.NET2003 参照の追加

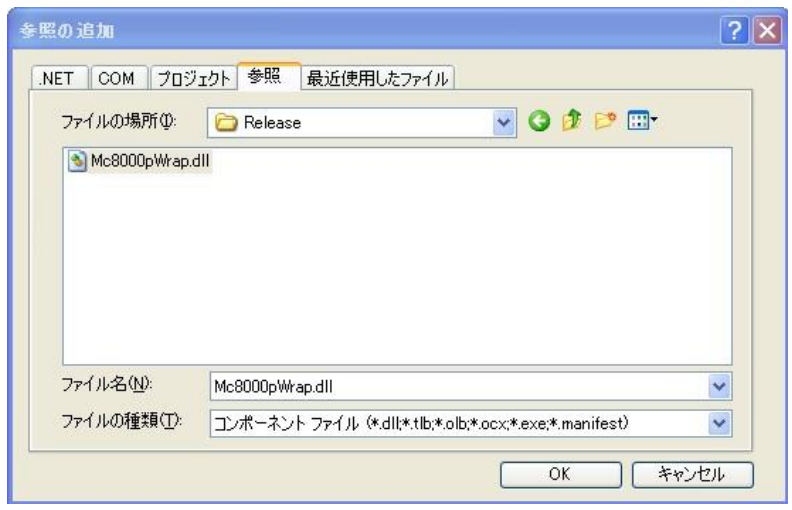


図 3.3-6 C# 2005、C# 2008 参照の追加

3.4　ＡＰＩ

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

3.4.1　関数一覧

下表は、API関数の一覧表です。
「VC」「VB」「VB.NET」「C#.NET」の欄は、各言語において各関数が使用できるかどうかを記載しています。
VC++、VC++.NETの場合　・・・・・・・・[VC] の項目を参照して下さい。
VB6.0の場合 ・・・・・・・・・・・・・[VB] の項目を参照して下さい。
VB.NET2003、VB2005、VB2008の場合・・[VB.NET] の項目が有る場合は[VB.NET]を、無い場合は[VB] の項目を参照して下さい。
C#.NETの場合　・・・・・・・・・・・・[C#.NET] と [C#] の項目を参照して下さい。

○は使用できます。×は使用できません。

（１）基本関数

関数名	説明	VC	VB	VB.NET	C#.NET	ページ	備考
Nmc_Open	ボードの使用を開始する	○	○	○	○	50	
Nmc_Close	ボードの使用を終了する	○	○	○	○	〃	
Nmc_CloseAll	全てのボードの使用を終了する	○	○	○	○	〃	
Nmc_GetBoardInfo	オープンしたボードの情報を取得する	○	○	○	○	51	
Nmc_OutPort	出力ポートにデータを書く	○	○	○	○	〃	
Nmc_InPort	入力ポートからデータを読む	○	○	○	○	〃	
Nmc_WriteReg	ボードのレジスタにデータを書き込む	○	○	○	○	52	
Nmc_ReadReg	ボードのレジスタからデータを読み込む	○	○	○	○	〃	
Nmc_SetEvent	割込みを処理するユーザー関数を設定する	○	×	×	○	53	
Nmc_ResetEvent	割込みを処理するユーザー関数の設定を解除する	○	×	×	○	54	
Nmc_ReadEvent	割込み発生直後の各軸ＲＲ３の値を取得する	○	×	×	○	〃	

（２）リセット、命令

関数名	説明	VC	VB	VB.NET	C#.NET	ページ	備考
Nmc_Reset	ボードに搭載しているＩＣをリセットする	○	○	○	○	55	
Nmc_Command	指定軸の命令を実行する	○	○	○	○	〃	
Nmc_Command_IP	補間命令を実行する	○	○	○	○	〃	※１
Nmc_WriteReg0	ＷＲ０(コマンドレジスタ)　書き込み	○	○	○	○	56	
Nmc_WriteReg1	ＷＲ１(モードレジスタ１)　書き込み	○	○	○	○	〃	
Nmc_WriteReg2	ＷＲ２(モードレジスタ２)　書き込み	○	○	○	○	〃	
Nmc_WriteReg3	ＷＲ３(モードレジスタ３)　書き込み	○	○	○	○	57	
Nmc_WriteReg4	ＷＲ４(アウトプットレジスタ)　書き込み	○	○	○	○	〃	
Nmc_WriteReg5	ＷＲ５　書き込み	○	○	○	○	〃	
Nmc_WriteReg6	ＷＲ６(ライトデータレジスタ１)書き込み	○	○	○	○	58	
Nmc_WriteReg7	ＷＲ７(ライトデータレジスタ２)書き込み	○	○	○	○	〃	

（４）リードレジスタ

関数名	説明	VC	VB	VB.NET	C#.NET	ページ	備考
Nmc_ReadReg0	ＲＲ０(主ステータスレジスタ)　読み出し	○	○	○	○	58	
Nmc_ReadReg1	ＲＲ１(ステータスレジスタ１)　読み出し	○	○	○	○	59	
Nmc_ReadReg2	ＲＲ２(ステータスレジスタ２)　読み出し	○	○	○	○	〃	
Nmc_ReadReg4	ＲＲ４(インプットレジスタ１)　読み出し	○	○	○	○	〃	
Nmc_ReadReg5	ＲＲ５(インプットレジスタ２)　読み出し	○	○	○	○	60	
Nmc_ReadReg6	ＲＲ６(リードデータレジスタ１)読み出し	○	○	○	○	〃	
Nmc_ReadReg7	ＲＲ７(リードデータレジスタ２)読み出し	○	○	○	○	〃	

（５）パラメータ設定

関数名	説明	VC	VB	VB.NET	C#.NET	ページ	備考
Nmc_Range	レンジ設定	○	○	○	○	61	
Nmc_Jerk	加速度増加率設定（加加速度）	○	○	○	○	〃	
Nmc_Acc	加速度設定	○	○	○	○	〃	
Nmc_Dec	減速度設定	○	○	○	○	62	
Nmc_StartSpd	初速度設定	○	○	○	○	〃	
Nmc_Speed	ドライブ速度設定	○	○	○	○	〃	
Nmc_Pulse	出力パルス数／補間終点設定(VC,C#用)	○	×	×	○	63	
Nmc_Pulse_VB	出力パルス数／補間終点設定(VB用)	×	○	○	×	〃	
Nmc_DecP	マニュアル減速点設定(VC,C#用)	○	×	×	○	64	
Nmc_DecP_VB	マニュアル減速点設定(VB用)	×	○	○	×	〃	
Nmc_Center	円弧中心点設定	○	○	○	○	〃	※１
Nmc_Lp	論理位置カウンタ設定	○	○	○	○	65	
Nmc_Ep	実位置カウンタ設定	○	○	○	○	〃	
Nmc_CompP	ＣＯＭＰ＋レジスタ設定	○	○	○	○	〃	
Nmc_CompM	ＣＯＭＰ－レジスタ設定	○	○	○	○	66	
Nmc_AccOfst	加速カウンタオフセット設定	○	○	○	○	〃	
Nmc_DJerk	減速度増加率設定	○	○	○	○	〃	※１
Nmc_HomeSpd	原点検出速度設定	○	○	○	○	67	

（６）その他のモード設定

関数名	説明	VC	VB	VB.NET	C#.NET	ページ	備考
Nmc_ExpMode	拡張モード設定	○	○	○	○	67	※１
Nmc_SyncMode	同期動作モード設定	○	○	○	○	68	※１
Nmc_HomeMode	自動原点出しモード設定	○	○	○	○	〃	※２

（７）データ読み出し

関数名	説明	VC	VB	VB.NET	C#.NET	ページ	備考
Nmc_ReadLp	論理位置カウンタ読み出し	○	○	○	○	69	
Nmc_ReadEp	実位置カウンタ読み出し	○	○	○	○	〃	
Nmc_ReadSpeed	現在ドライブ速度読み出し	○	○	○	○	〃	
Nmc_ReadAccDec	現在加／減速度読み出し	○	○	○	○	70	
Nmc_ReadSyncBuff	同期バッファレジスタ読み出し	○	○	○	○	〃	※１

（８）状態取得

関数名	説明	VC	VB	VB.NET	C#.NET	ページ	備考
Nmc_GetDriveStatus	ドライブ状態取得	○	○	○	○	71	
Nmc_GetCNextStatus	連続補間次データ書込み可能状態取得	○	○	○	○	〃	※１
Nmc_GetBpSc	ＢＰ補間スタックカウンタ取得	○	○	○	○	72	※１

（９）書き込み・読み出し

関数名	説明	VC	VB	VB.NET	C#.NET	ページ	備考
Nmc_WriteRegSetAxis	軸指定ライトレジスタ書き込み(WR1～3)	○	○	○	○	72	
Nmc_ReadRegSetAxis	軸指定リードレジスタ読み出し(RR1～2)	○	○	○	○	73	
Nmc_WriteData	データ書き込み（パラメータ）	○	○	○	○	〃	
Nmc_WriteData2	データ書き込み（拡張モード、同期動作モード）	○	○	○	○	74	※１
Nmc_ReadData	データ読み出し	○	○	○	○	〃	

（１０）補間実行

関数名	説明	VC	VB	VB.NET	C#.NET	ページ	備考
Nmc_2BPExec	２軸ＢＰ補間実行	○	○	○	○	75	※１
Nmc_3BPExec	３軸ＢＰ補間実行	○	○	○	○	77	※１
Nmc_2BPExec_BG	２軸ＢＰ補間実行（バックグラウンドで実行）	○	○	○	○	79	※１
Nmc_3BPExec_BG	３軸ＢＰ補間実行（バックグラウンドで実行）	○	○	○	○	82	※１
Nmc_2CIPExec	２軸連続補間実行	○	○	○	○	85	※１
Nmc_3CIPExec	３軸連続補間実行	○	○	○	○	87	※１
Nmc_2CIPExec_BG	２軸連続補間実行（バックグラウンドで実行）	○	○	○	○	89	※１
Nmc_3CIPExec_BG	３軸連続補間実行（バックグラウンドで実行）	○	○	○	○	92	※１
Nmc_IPStop	補間実行を中断する	○	○	○	○	95	※１
Nmc_IPGetMsgNo	補間終了時の受信ﾒｯｾｰｼﾞからﾎﾞｰﾄﾞ番号とＩＣ番号取得	○	○	○	○	〃	※１

※１：MCX314As 専用関数

※２：MCX304 専用関数

3.4.2 関数仕様

VC++、VC++.NETの場合　　：**VC** と[VC]の項目を参照して下さい。
VB6.0の場合　　：**VB** と[VB]の項目を参照して下さい。
VB.NET2003、VB2005、VB2008の場合　　：**VB.NET** と[VB]または[VB.NET]の項目を参照して下さい。
C#.NETの場合　　：**C#.NET** と[C#]の項目を参照して下さい。
指定のない項目は、各言語共通の内容です。

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

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_GetBoardInfo</td><td>
オープンしたボードの情報としてデバイスＩＤを取得する。

VC　　BOOL　　Nmc_GetBoardInfo(int No, USHORT* DeviceID);
VB　　Function Nmc_GetBoardInfo(ByVal No As Long, ByRef DeviceID As Integer) As Long
VB.NET　Function Nmc_GetBoardInfo(ByVal No As Integer, ByRef DeviceID As Short) As Integer
C#.NET　bool　　MC8000P.Nmc_GetBoardInfo(int No, out ushort DeviceID);
入力パラメータ
　No　　ボード番号（ボード上のロータリースイッチの値(0～15)）
　DeviceID　[VC] 取得したボードのデバイスＩＤを格納する変数のアドレス
　　[VB][C#] 取得したボードのデバイスＩＤを格納する変数
　　各ボードのデバイスＩＤは補足説明(1)③参照。
　　[VC][VB]　MC8082P,42P,22PはID_MC8082P、MC8080PはID_MC8080P、MC8043PはID_MC8043P。
　　[C#]　　MC8082P,42P,22PはDev_ID.MC8082P、MC8080PはDev_ID.MC8080P、MC8043PはDev_ID.MC8043Pを指定する。
戻り値
　[VC]　取得が成功するとTRUE、失敗するとFALSE
　[VB]　取得が成功すると０以外、失敗すると０
　[C#]　取得が成功するとtrue、失敗するとfalse
使用例
　[VC]　　USHORT DeviceID;
　　status = Nmc_GetBoardInfo(No, &DeviceID);　// デバイスＩＤ取得
　　if(DeviceID == ID_MC8082P)　　// MC8082P,42P,22Pの場合
　[VB]　　Dim DeviceID As Integer
　　status = Nmc_GetBoardInfo(No, DeviceID)
　　If DeviceID = ID_MC8082P Then
　[VB.NET]　Dim DeviceID As Short
　　status = Nmc_GetBoardInfo(No, DeviceID)
　　If DeviceID = ID_MC8082P Then
　[C#]　　ushort DeviceID;
　　status = Nmc_GetBoardInfo(No, out DeviceID);　// デバイスＩＤ取得
　　if(DeviceID == Dev_ID.MC8082P)　　// MC8082P,42P,22Pの場合
</td></tr>
<tr><td>Nmc_OutPort</td><td>
出力ポートに２バイトデータを書き込む。

VC　　void Nmc_OutPort(int No, long Adr, long Data);
VB　　Sub　Nmc_OutPort(ByVal No As Long, ByVal Adr As Long, ByVal Data As Long)
VB.NET　Sub　Nmc_OutPort(ByVal No As Integer, ByVal Adr As Integer, ByVal Data As Integer)
C#.NET　void MC8000P.Nmc_OutPort(int No, REG_MCX Adr, int Dat);
入力パラメータ
　No　　ボード番号（ボード上のロータリースイッチの値(0～15)）
　Adr　　書き込むアドレス。各ボード取扱説明書に記載しているＩ／Ｏアドレス。
　　詳細は補足説明(1)①、(6)参照。
　Data　書き込むデータ
戻り値
　なし
使用例
　[VC]　Nmc_OutPort(No, 0, 0x8000);　　// ＩＣ－Ａのソフトリセット(WR0書き込み)
　[VB]　Call Nmc_OutPort(No, 0, &H8000)
　[C#]　MC8000P.Nmc_OutPort(No, REG_MCX.WR0_A, 0x8000);
</td></tr>
<tr><td>Nmc_InPort</td><td>
入力ポートから２バイトデータを読み出す。

VC　　long　　Nmc_InPort(int No, long Adr);
VB　　Function Nmc_InPort(ByVal No As Long, ByVal adr As Long) As Long
VB.NET　Function Nmc_InPort(ByVal No As Integer, ByVal adr As Integer) As Integer
C#.NET　int　　MC8000P.Nmc_InPort(int No, REG_MCX Adr);
入力パラメータ
　No　　ボード番号（ボード上のロータリースイッチの値(0～15)）
　Adr　　読み出すアドレス。各ボード取扱説明書に記載しているＩ／Ｏアドレス。
　　詳細は補足説明(1)①、(6)参照。
戻り値
　入力ポートから読み込んだデータ
使用例
　[VC]　data = Nmc_InPort(No, 0);　　// ＩＣ－Ａのリードレジスタ RR0 の読み出し
　[VB]　data = Nmc_InPort(No, 0)
　[C#]　data = MC8000P.Nmc_InPort(No, REG_MCX.RR0_A);
注意
　[VC]　ＲＲ３レジスタのデータ読み出しに関しては、Nmc_ReadEvent関数の説明を参考にして下さい。
</td></tr>
</table>

関数名	機能　及び　内容
Nmc_WriteReg	ボードのライトレジスタ(WR0～WR7)にデータを書き込む。 **VC**　　void Nmc_WriteReg(int No, int IcNo, long RegNum, long Dat); **VB**　　Sub Nmc_WriteReg(ByVal No As Long, ByVal IcNo As Long, ByVal RegNum As Long, ByVal Dat As Long) **VB.NET**　Sub Nmc_WriteReg(ByVal No As Integer, ByVal IcNo As Integer, ByVal RegNum As Integer, ByVal Dat As Integer) **C#.NET**　void MC8000P.Nmc_WriteReg(int No, int IcNo, int RegNum, int Dat); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 RegNum　書き込むレジスタ(MCX_WR0～MCX_WR7)。詳細は補足説明(1)参照。 例）[VC][VB]WR0の場合は MCX_WR0 を、WR1の場合は MCX_WR1 を指定する。 [C#]WR0の場合は REG_MCX.WR0 を、WR1の場合は REG_MCX.WR1 を指定する。 Dat　　書き込むデータ **戻り値** なし **使用例** [VC]　Nmc_WriteReg(No, 0, MCX_WR0, 0x8000);　　// ＩＣ－Ａのソフトリセット(WR0書き込み) [VB]　Call Nmc_WriteReg(No, 0, MCX_WR0, &H8000) [C#]　MC8000P.Nmc_WriteReg(No, 0, REG_MCX.WR0, 0x8000);
Nmc_ReadReg	ボードのリードレジスタ(RR0～RR7)からデータを読み出す。 **VC**　　long Nmc_ReadReg(int No, int IcNo, long RegNum); **VB**　　Function Nmc_ReadReg(ByVal No As Long, ByVal IcNo As Long, ByVal RegNum As Long) As Long **VB.NET**　Function Nmc_ReadReg(ByVal No As Integer, ByVal IcNo As Integer, ByVal RegNum As Integer) As Integer **C#.NET**　void MC8000P.Nmc_ReadReg(int No, int IcNo, int RegNum); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 RegNum　読み出すレジスタ(MCX_RR0～MCX_RR7)。詳細は補足説明(1)参照。 例）[VC][VB]RR0の場合は MCX_RR0 を、RR1の場合は MCX_RR1 を指定する。 [C#]RR0の場合は REG_MCX.RR0 を、RR1の場合は REG_MCX.RR1 を指定する。 **戻り値** リードレジスタから読み出したデータ **使用例** [VC]　data = Nmc_ReadReg(No, 0, MCX_RR0);　// ＩＣ－Ａのリードレジスタ RR0 の読み出し [VB]　data = Nmc_ReadReg(No, 0, MCX_RR0) [C#]　data = MC8000P.Nmc_ReadReg(No, 0, REG_MCX.WR0); **注意** [VC]　ＲＲ３レジスタのデータ読み出しに関しては、Nmc_ReadEvent関数の説明を参考にして下さい。

関数名	機能　及び　内容

Nmc_SetEvent	割込みを処理するユーザー関数を設定する。
	この関数を実行すると、割り込みが発生した時にユーザー関数が呼び出され、指定した引数が１つ渡されます(C#は、引数を指定できません)。このユーザー関数は１つのスレッドとして起動されます。
	割込み処理する関数の設定を解除する場合は Nmc_ResetEvent を実行して下さい。
	
	VC　　BOOL Nmc_SetEvent(int No, LPTHREAD_START_ROUTINE UserThread, LPVOID lpParameter);
	VB　　使用できません
	VB.NET　使用できません
	C#.NET　bool MC8000P.Nmc_SetEvent(int No, UserThread Callback, int param);
	
	入力パラメータ
	No　　ボード番号（ボード上のロータリースイッチの値(0～15)）
	[VC] UserThread　ユーザー関数のアドレス
	[VC] lpParameter　ユーザー関数スレッドに渡す１つの引数を指定する。
	スレッドで使用可能なポインタを設定して下さい。
	引数を使用しない場合は、NULL等で良い。
	ポインタの場合、ユーザー関数呼び出し時に使用可能なポインタを
	設定して下さい。
	[C#] Callback　ユーザーメソッド（デリゲート型）
	詳細は3.4.3使用方法を参照。
	param　割込み発生時にユーザー関数に渡す１つのパラメータ(数値等int限定)を指定します。
	
	戻り値
	[VC] 成功するとTRUE、失敗するとFALSE
	[C#] 成功するとtrue、失敗するとfalse
	
	使用例
	[VC]
	（ボード番号０の場合）
	status = Nmc_SetEvent(0, MC_EventFunc0, lpParam); //　関数のアドレスと引数を設定
	Nmc_WriteReg1(0, 0, AXIS_ALL, 0x8000);　　//　ＩＣ－Ａの停止時割込み発生（全軸）
	
	（ボード番号１の場合）
	status = Nmc_SetEvent(1, MC_EventFunc1, NULL);　　//　関数のアドレスと引数を設定
	Nmc_WriteReg1(1, 0, AXIS_ALL, 0x8000);　　//　ＩＣ－Ａの停止時割込み発生（全軸）
	
	■割込みユーザー関数例
	DWORD WINAPI MC_EventFunc0(LPVOID lpParam)
	{
	long Rr3X, Rr3Y, Rr3Z, Rr3U;
	Nmc_ReadEvent(0, 0, &Rr3X,&Rr3Y,&Rr3Z,&Rr3U); // ﾎﾞｰﾄﾞ0,IC-AのRR3割込みデータ読み出し
	・・・・
	return 0;
	}
	
	DWORD WINAPI MC_EventFunc1(LPVOID lpParam)
	{
	long Rr3X, Rr3Y, Rr3Z, Rr3U;
	Nmc_ReadEvent(1, 0, &Rr3X,&Rr3Y,&Rr3Z,&Rr3U); // ﾎﾞｰﾄﾞ1,IC-AのRR3割込みデータ読み出し
	・・・・
	return 0;
	}
	[C#]
	// 割込みユーザーメソッド isr をデリゲート型変数に代入
	MC8000P.callback[0] = new MC8000P.UserThread(isr);
	// 割込みユーザーメソッドの設定
	MC8000P.Nmc_SetEvent(no, MC8000P.callback[0], param);

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_ResetEvent</td><td>割込みを処理するユーザー関数の設定を解除する。
この関数を実行すると、割り込みが発生してもユーザー関数は呼び出されません。

VC　　BOOL Nmc_ResetEvent(int No);
VB　　使用できません
VB.NET　使用できません
C#.NET　bool MC8000P.Nmc_ResetEvent(int No);

入力パラメータ
　No　　ボード番号（ボード上のロータリースイッチの値(0～15)）

戻り値
　[VC]　成功するとTRUE、失敗するとFALSE
　[C#]　成功するとtrue、失敗するとfalse

使用例
　[VC]　status = Nmc_ResetEvent(No);
　[C#]　MC8000P.Nmc_WriteReg1(No, AXIS.ALL, 0x0000);　　//　割込み禁止（全軸）
　　status = MC8000P.Nmc_ResetEvent(No);</td></tr>
<tr><td>Nmc_ReadEvent</td><td>割込み発生直後の各軸ＲＲ３の値を取得する。（ドライバ内のＲＲ３データは読み出し後クリアされる）

VC　　BOOL Nmc_ReadEvent(int No, int IcNo, long* Rr3X, long* Rr3Y, long* Rr3Z, long* Rr3U);
VB　　使用できません
VB.NET　使用できません
C#.NET　bool MC8000P.Nmc_ReadEvent(int No, int IcNo, out int Rr3X, out int Rr3Y, out int Rr3Z, out int Rr3U);
入力パラメータ
　No　　ボード番号（ボード上のロータリースイッチの値(0～15)）
　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。
　Rr3X　Ｘ軸のＲＲ３を格納する為のバッファへのポインタ
　Rr3Y　Ｙ軸のＲＲ３を格納する為のバッファへのポインタ
　Rr3Z　Ｚ軸のＲＲ３を格納する為のバッファへのポインタ
　Rr3U　Ｕ軸のＲＲ３を格納する為のバッファへのポインタ

戻り値
　[VC]　成功するとTRUE、失敗するとFALSE
　[C#]　成功するとtrue、失敗するとfalse

使用例
　[VC]　long Rr3X[2], Rr3Y[2], Rr3Z[2], Rr3U[2];
　　Nmc_ReadEvent(No, 0, &Rr3X[0], &Rr3Y[0], &Rr3Z[0], &Rr3U[0]); //IC-AのRR3データを読み出す
　　Nmc_ReadEvent(No, 1, &Rr3X[1], &Rr3Y[1], &Rr3Z[1], &Rr3U[1]); //IC-BのRR3データを読み出す
　[C#]　int Rr3X,Rr3Y,Rr3Z,Rr3U ;　　// Ｘ軸、Ｙ軸、Ｚ軸、Ｕ軸
　　MC8000P.Nmc_ReadEvent(No, IcNo, out Rr3X, out Rr3Y, out Rr3Z, out Rr3U);

注意
　ボードで割込が発生した直後ドライバ内でＲＲ３を読み出してしまうのでＲＲ３はクリアされてしまいます。割込み発生直後のＲＲ３を確認する場合はこの関数を使用してください。

　また、Nmc_SetEvent、Nmc_ResetEvent関数の実行とは関係なく、割込みが発生するとドライバは必ずRR3データを読み出し保存します。ドライバ内に保存されたRR3データは、Nmc_ReadEvent関数を実行して読み出すとクリアされます。
　ドライバ内のRR3データをクリアしたい時は、Nmc_ReadEvent関数を実行して下さい。</td></tr>
</table>

関数名	機能　及び　内容
Nmc_Reset	ボードに搭載しているＩＣをリセットする。 **VC**　　void Nmc_Reset(int No, int IcNo); **VB**　　Sub Nmc_Reset(ByVal No As Long, ByVal IcNo As Long) **VB.NET**　Sub Nmc_Reset(ByVal No As Integer, ByVal IcNo As Integer) **C#.NET**　void MC8000P.Nmc_Reset(int No, int IcNo); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 **戻り値** なし **使用例** [VC]　Nmc_Reset(1, 0);　　// ボード番号１のＩＣ－Ａをリセットする [VB]　Call Nmc_Reset(1, 0) [C#]　MC8000P.Nmc_Reset(1, 0);
Nmc_Command	指定軸の命令を実行する。（WR0に指定軸の命令を書く） **VC**　　void Nmc_Command(int No, int IcNo, int Axis, int cmd); **VB**　　Sub Nmc_Command(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long,ByVal cmd As Long)**VB.NET**　Sub Nmc_Command(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal cmd As Integer) **C#.NET**　void MC8000P.Nmc_Command(int No, int IcNo, AXIS Axis, CMD cmd); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　命令を実行する軸。複数軸指定可能。詳細は補足説明(2)参照。 cmd　命令番号。定義ファイル(※１)内のコマンド定義の「ドライブ命令、その他の命令」の中から一つを指定する。＋方向定量ドライブの場合はCMD_F_DRV_Pを指定する。C#は補足説明(4)参照。 ※１：[VC]MC8000P_DLL.H, [VB]MC8000P_DLL.bas, [VB.NET]MC8000P_DLL.vb **戻り値** なし **使用例** [VC]　Nmc_Command(No, IcNo, AXIS_X, CMD_F_DRV_P);　　// Ｘ軸の＋方向定量ドライブを実行する [VB]　Call Nmc_Command(No, IcNo, AXIS_X, CMD_F_DRV_P) [C#]　MC8000P.Nmc_Command(No, IcNo, AXIS.X, CMD.CMD_F_DRV_P);
Nmc_Command_IP	補間命令を実行する。（WR0に補間命令を書く）　※MCX314As専用 **VC**　　void Nmc_Command_IP(int No, int IcNo, int cmd); **VB**　　Sub Nmc_Command_IP(ByVal No As Long, ByVal IcNo As Long, ByVal cmd As Long) **VB.NET**　Sub Nmc_Command_IP(ByVal No As Integer, ByVal IcNo As Integer, ByVal cmd As Integer) **C#.NET**　void MC8000P.Nmc_Command_IP(int No, int IcNo, CMD cmd); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 cmd　命令番号。定義ファイル(※１)内のコマンド定義の「補間命令」の中から一つを指定する。 ２軸直線補間ドライブの場合はCMD_IP_2STを指定する。C#は補足説明(4)参照。 ※１：[VC]MC8000P_DLL.H, [VB]MC8000P_DLL.bas, [VB.NET]MC8000P_DLL.vb **戻り値** なし **使用例** [VC]　Nmc_WriteReg5(No, IcNo, 0x0004);　　// 補間軸設定（主軸：Ｘ、第２軸：Ｙ） Nmc_Command_IP(No, IcNo, CMD_IP_2ST);　　// ２軸直線補間ドライブを実行する [VB]　Call Nmc_WriteReg5(No, IcNo, &H0004)　　'　補間軸設定（主軸：Ｘ、第２軸：Ｙ） Call Nmc_Command_IP(No, IcNo, CMD_IP_2ST)　'　２軸直線補間ドライブを実行する [C#]　MC8000P.Nmc_WriteReg5(No, IcNo, 0x0004);　// 補間軸設定（主軸：Ｘ、第２軸：Ｙ） MC8000P.Nmc_Command_IP(No, IcNo, CMD.CMD_IP_2ST);　// ２軸直線補間ドライブを実行する

関数名	機能　及び　内容
Nmc_WriteReg0	ＷＲ０(コマンドレジスタ)にデータを書き込む。 **VC**　　void Nmc_WriteReg0(int No, int IcNo, long wdata); **VB**　　Sub Nmc_WriteReg0(ByVal No As Long, ByVal IcNo As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg0(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_WriteReg0(int No, int IcNo, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0，2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 wdata　書き込むデータ **戻り値** なし **使用例** [VC]　Nmc_WriteReg0(No, IcNo, 0x0120);　　// Ｘ軸の＋方向定量ドライブを実行する [VB]　Call Nmc_WriteReg0(No, IcNo, &H120) [C#]　MC8000P.Nmc_WriteReg0(No, IcNo, 0x0120);
Nmc_WriteReg1	ＷＲ１(モードレジスタ１)にデータを書き込む。 **VC**　　void Nmc_WriteReg1(int No, int IcNo, int Axis, long wdata); **VB**　　Sub Nmc_WriteReg1(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg1(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_WriteReg1(int No, int IcNo, AXIS Axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0，2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。詳細は補足説明(2)参照。 wdata　書き込むデータ **戻り値** なし **使用例** [VC]　Nmc_WriteReg1(No, IcNo, AXIS_X, 0x8000);　　// 停止時割り込み発生(Ｘ軸) [VB]　Call Nmc_WriteReg1(No, IcNo, AXIS_X, &H8000) [C#]　MC8000P.Nmc_WriteReg1(No, IcNo, AXIS.X, 0x8000);
Nmc_WriteReg2	ＷＲ２(モードレジスタ２)にデータを書き込む。 **VC**　　void Nmc_WriteReg2(int No, int IcNo, int Axis, long wdata); **VB**　　Sub Nmc_WriteReg2(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg2(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_WriteReg2(int No, int IcNo, AXIS Axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0，2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。詳細は補足説明(2)参照。 wdata　書き込むデータ **戻り値** なし **使用例** [VC]　Nmc_WriteReg2(No, IcNo, AXIS_Y, 0x2000);　　// ALARM 有効(Ｙ軸) [VB]　Call Nmc_WriteReg2(No, IcNo, AXIS_Y, &H2000) [C#]　MC8000P.Nmc_WriteReg2(No, IcNo, AXIS.Y, 0x2000);

関数名	機能　及び　内容
Nmc_WriteReg3	ＷＲ３(モードレジスタ３)にデータを書き込む。 **VC**　　void Nmc_WriteReg3(int No, int IcNo, int Axis, long wdata); **VB**　　Sub Nmc_WriteReg3(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg3(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_WriteReg3(int No, int IcNo, AXIS Axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを書き込む軸。AXIS_X, AXIS_Y 等を指定。複数軸指定可能。詳細は補足説明(2)参照。 wdata　書き込むデータ **戻り値** なし **使用例** [VC]　Nmc_WriteReg3(No, IcNo, AXIS_ALL, 0x0001);　　// 全軸マニュアル減速 [VB]　Call Nmc_WriteReg3(No, IcNo, AXIS_ALL, &H1) [C#]　MC8000P.Nmc_WriteReg3(No, IcNo, AXIS.ALL, 0x0001);
Nmc_WriteReg4	ＷＲ４(アウトプットレジスタ)にデータを書き込む。 **VC**　　void Nmc_WriteReg4(int No, int IcNo, long wdata); **VB**　　Sub Nmc_WriteReg4(ByVal No As Long, ByVal IcNo As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg4(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_WriteReg4(int No, int IcNo, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 wdata　書き込むデータ **戻り値** なし **使用例** [VC]　Nmc_WriteReg4(No, IcNo, 0x0001);　　// Ｘ軸汎用出力OUT0 Hiレベル出力 [VB]　Call Nmc_WriteReg4(No, IcNo, &H0001) [C#]　MC8000P.Nmc_WriteReg4(No, IcNo, 0x0001);
Nmc_WriteReg5	ＷＲ５にデータを書き込む。 **VC**　　void Nmc_WriteReg5(int No, int IcNo, long wdata); **VB**　　Sub Nmc_WriteReg5(ByVal No As Long, ByVal IcNo As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_WriteReg5(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_WriteReg5(int No, int IcNo, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 wdata　書き込むデータ **戻り値** なし **使用例** **■MCX304** [VC]　Nmc_WriteReg5(No, IcNo, 0x0001);　　// Ｘ軸汎用出力OUT0 有効 [VB]　Call Nmc_WriteReg5(No, IcNo, &H1) [C#]　MC8000P.Nmc_WriteReg5(No, IcNo, 0x0001); **■MCX314As** [VC]　Nmc_WriteReg5(No, IcNo, 0x0024);　　// 補間軸設定(主軸：Ｘ、第２軸：Ｙ、第３軸：Ｚ) [VB]　Call Nmc_WriteReg5(No, IcNo, &H0024) [C#]　MC8000P.Nmc_WriteReg5(No, IcNo, 0x0024);

関数名	機能　及び　内容
Nmc_WriteReg6	ＷＲ６(ライトデータレジスタ１)にデータを書き込む。 **VC**　　void Nmc_WriteReg6(int No, int IcNo, long wdata); **VB**　　Sub　Nmc_WriteReg6(ByVal No As Long, ByVal IcNo As Long, ByVal wdata As Long) **VB.NET**　Sub　Nmc_WriteReg6(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_WriteReg6(int No, int IcNo, int wdata); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～15)） 　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 　wdata　書き込むデータ **戻り値** 　なし **使用例** 　[VC]　Nmc_WriteReg6(No, IcNo, 0x1234);　　// ライトデータレジスタ１にデータ(1234)Hを書く 　[VB]　Call Nmc_WriteReg6(No, IcNo, &H1234) 　[C#]　MC8000P.Nmc_WriteReg6(No, IcNo, 0x1234);
Nmc_WriteReg7	ＷＲ７(ライトデータレジスタ２)にデータを書き込む。 **VC**　　void Nmc_WriteReg7(int No, int IcNo, long wdata); **VB**　　Sub　Nmc_WriteReg7(ByVal No As Long, ByVal IcNo As Long, ByVal wdata As Long) **VB.NET**　Sub　Nmc_WriteReg7(ByVal No As Integer, ByVal IcNo As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_WriteReg7(int No, int IcNo, int wdata); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～15)） 　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 　wdata　書き込むデータ **戻り値** 　なし **使用例** 　[VC]　Nmc_WriteReg7(No, IcNo, 0x5678);　　// ライトデータレジスタ２にデータ(5678)Hを書く 　[VB]　Call Nmc_WriteReg7(No, IcNo, &H5678) 　[C#]　MC8000P.Nmc_WriteReg7(No, IcNo, 0x5678);
Nmc_ReadReg0	ＲＲ０(主ステータスレジスタ)のデータを読み出す。 **VC**　　long　　Nmc_ReadReg0(int No, int IcNo); **VB**　　Function Nmc_ReadReg0(ByVal No As Long, ByVal IcNo As Long) As Long **VB.NET**　Function Nmc_ReadReg0(ByVal No As Integer, ByVal IcNo As Integer) As Integer **C#.NET**　int　　MC8000P.Nmc_ReadReg0(int No, int IcNo); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～15)） 　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 **戻り値** 　ＲＲ０(主ステータスレジスタ)のデータ **使用例** 　[VC]　Data = Nmc_ReadReg0(No, IcNo);　　// ＲＲ０のデータを読む 　[VB]　Data = Nmc_ReadReg0(No, IcNo) 　[C#]　Data = MC8000P.Nmc_ReadReg0(No, IcNo);

関数名	機能　及び　内容
Nmc_ReadReg1	ＲＲ１(ステータスレジスタ１)のデータを読み出す。 **VC**　long　Nmc_ReadReg1(int No, int IcNo, int Axis); **VB**　Function Nmc_ReadReg1(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadReg1(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　MC8000P.Nmc_ReadReg1(int No, int IcNo, AXIS Axis); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを読み出す軸。Ｘ軸はAXIS_X, Ｙ軸はAXIS_Y, Ｚ軸はAXIS_Z, Ｕ軸はAXIS_Uを指定する。詳細は補足説明(2)参照。 **戻り値** ＲＲ１(ステータスレジスタ１)のデータ **使用例** [VC]　Data = Nmc_ReadReg1(No, IcNo, AXIS_X);　　// Ｘ軸のＲＲ１のデータを読む [VB]　Data = Nmc_ReadReg1(No, IcNo, AXIS_X) [C#]　Data = MC8000P.Nmc_ReadReg1(No, IcNo, AXIS.X);
Nmc_ReadReg2	ＲＲ２(ステータスレジスタ２)のデータを読み出す。 **VC**　long　Nmc_ReadReg2(int No, int IcNo, int Axis); **VB**　Function Nmc_ReadReg2(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadReg2(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　MC8000.Nmc_ReadReg2(int No, int IcNo, AXIS Axis); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを読み出す軸。Ｘ軸はAXIS_X, Ｙ軸はAXIS_Y, Ｚ軸はAXIS_Z, Ｕ軸はAXIS_Uを指定する。詳細は補足説明(2)参照。 **戻り値** ＲＲ２(ステータスレジスタ２)のデータ **使用例** [VC]　Data = Nmc_ReadReg2(No, IcNo, AXIS_Y);　　// Ｙ軸のＲＲ２のデータを読む [VB]　Data = Nmc_ReadReg2(No, IcNo, AXIS_Y) [C#]　Data = MC8000.Nmc_ReadReg2(No, IcNo, AXIS.Y);
Nmc_ReadReg4	ＲＲ４(インプットレジスタ１)のデータを読み出す。 **VC**　long　Nmc_ReadReg4(int No, int IcNo); **VB**　Function Nmc_ReadReg4(ByVal No As Long, ByVal IcNo As Long) As Long **VB.NET**　Function Nmc_ReadReg4(ByVal No As Integer, ByVal IcNo As Integer) As Integer **C#.NET**　int　MC8000P.Nmc_ReadReg4(int No, int IcNo); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 **戻り値** ＲＲ４(インプットレジスタ１)のデータ **使用例** [VC]　Data = Nmc_ReadReg4(No, IcNo);　　// ＲＲ４のデータを読む [VB]　Data = Nmc_ReadReg4(No, IcNo) [C#]　Data = MC8000P.Nmc_ReadReg4(No, IcNo);

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

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

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

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_Pulse</td><td>出力パルス数、あるいは補間終点を設定する。(VC専用)　※MCX304には補間機能がありません。

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

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

入力パラメータ
　No　　ボード番号（ボード上のロータリースイッチの値(0～15)）
　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。
　Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。
　wdata　設定するデータ

戻り値
　なし

使用例
　[VC]　Nmc_Pulse(No, IcNo, AXIS_X, 2000);　　// 出力パルス数に 2000 を設定する(Ｘ軸)

　　Nmc_Pulse(No, IcNo, AXIS_Y,　300);　　// 補間終点に　300 を設定する(Ｙ軸)
　　Nmc_Pulse(No, IcNo, AXIS_Z, -400);　　// 補間終点に -400 を設定する(Ｚ軸)
　[C#]　MC8000P.Nmc_Pulse(No, IcNo, AXIS.X, 2000);
　　MC8000P.Nmc_Pulse(No, IcNo, AXIS.Y, 300);
　　MC8000P.Nmc_Pulse(No, IcNo, AXIS.Z, -400);</td></tr>
<tr><td>Nmc_Pulse_VB</td><td>出力パルス数、あるいは補間終点を設定する。(VB専用)　※MCX304には補間機能がありません。

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

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

入力パラメータ
　No　　ボード番号（ボード上のロータリースイッチの値(0～15)）
　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。
　Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。
　wdata　設定するデータ

戻り値
　なし

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

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

関数名	機能　及び　内容
Nmc_DecP	マニュアル減速点を設定する。(VC,C#専用) **VC**　　void Nmc_DecP(int No, int IcNo, int Axis, ULONG wdata); **VB**　　使用できません **VB.NET**　使用できません **C#.NET**　void MC8000P.Nmc_DecP(int No, int IcNo, AXIS Axis, uint wdata); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～15)） 　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 　Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 　wdata　設定するデータ **戻り値** 　なし **使用例** 　[VC]　Nmc_DecP(No, IcNo, AXIS_U, 30000);　　// マニュアル減速点に 30000 を設定する(U軸) 　[C#]　MC8000P.Nmc_DecP(No, IcNo, AXIS.U, 30000);
Nmc_DecP_VB	マニュアル減速点を設定する。(VB専用) **VC**　　使用できません **VB**　　Sub Nmc_DecP_VB(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal wdata As Double) **VB.NET**　Sub Nmc_DecP_VB(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Double) **C#.NET**　使用できません **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～15)） 　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 　Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 　wdata　設定するデータ **戻り値** 　なし **使用例** 　[VB]　Call Nmc_DecP_VB(No, IcNo, AXIS_X, 40000)　' マニュアル減速点に 40000 を設定する(X軸)
Nmc_Center	円弧中心点を設定する。　※MCX314As専用 **VC**　　void Nmc_Center(int No, int IcNo, int Axis, long wdata); **VB**　　Sub　Nmc_Center(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub　Nmc_Center(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_Center(int No, int IcNo, AXIS Axis, int wdata); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～15)） 　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 　Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 　wdata　設定するデータ **戻り値** 　なし **使用例** 　[VC]　Nmc_Center(No, IcNo, AXIS_Y, 1500);　　// 円弧中心点に 1500 を設定する(Y軸) 　[VB]　Call Nmc_Center(No, IcNo, AXIS_Y, 1500) 　[C#]　MC8000P.Nmc_Center(No, IcNo, AXIS.Y, 1500);

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

関数名	機能　及び　内容
Nmc_CompM	ＣＯＭＰ－レジスタを設定する。 **VC**　　void Nmc_CompM(int No, int IcNo, int Axis, long wdata); **VB**　　Sub Nmc_CompM(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long,ByVal wdata As Long)**VB.NET** Sub Nmc_CompM(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET** void MC8000P.Nmc_CompM(int No, int IcNo, AXIS Axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC] Nmc_CompM(No, IcNo, AXIS_X, -50000);　　// ＣＯＭＰ－レジスタに -50000 を設定する(Ｘ軸) [VB] Call Nmc_CompM(No, IcNo, AXIS_X, -50000) [C#] MC8000P.Nmc_CompM(No, IcNo, AXIS.X, -50000);
Nmc_AccOfst	加速カウンタオフセットを設定する。 **VC**　　void Nmc_AccOfst(int No, int IcNo, int Axis, long wdata); **VB**　　Sub Nmc_AccOfst(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET** Sub Nmc_AccOfst(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET** void MC8000P.Nmc_AccOfst(int No, int IcNo, AXIS Axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC] Nmc_AccOfst(No, IcNo, AXIS_Y, 20);　　// 加速カウンタオフセットに 20 を設定する(Ｙ軸) [VB] Call Nmc_AccOfst(No, IcNo, AXIS_Y, 20) [C#] MC8000P.Nmc_AccOfst(No, IcNo, AXIS.Y, 20);
Nmc_DJerk	減速度増加率を設定する。　※MCX314As専用 **VC**　　void Nmc_DJerk(int No, int IcNo, int Axis, long wdata); **VB**　　Sub Nmc_DJerk(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long,ByVal wdata As Long)**VB.NET** Sub Nmc_DJerk(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET** void MC8000P.Nmc_DJerk(int No, int IcNo, AXIS Axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC] Nmc_DJerk(No, IcNo, AXIS_Z, 1000);　　// 減速度増加率に 1000 を設定する(Ｚ軸) [VB] Call Nmc_DJerk(No, IcNo, AXIS_Z, 1000) [C#] MC8000P.Nmc_DJerk(No, IcNo, AXIS.Z, 1000);

関数名	機能　及び　内容
Nmc_HomeSpd	原点検出速度を設定する。 **VC**　　void Nmc_HomeSpd(int No, int IcNo, int Axis, long wdata); **VB**　　Sub Nmc_HomeSpd(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal wdata As Long) **VB.NET**　Sub Nmc_HomeSpd(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_HomeSpd(int No, int IcNo, AXIS Axis, int wdata); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 wdata　設定するデータ **戻り値** なし **使用例** [VC] Nmc_HomeSpd(No, IcNo, AXIS_U, 200);　　// 原点検出速度に 200 を設定する(U軸) [VB] Call Nmc_HomeSpd(No, IcNo, AXIS_U, 200) [C#] MC8000P.Nmc_HomeSpd(No, IcNo, AXIS.U, 200);
Nmc_ExpMode	拡張モードを設定する。　※MCX314As専用 **VC**　　void Nmc_ExpMode(int No, int IcNo, int Axis, long EM6_data, long EM7_data); **VB**　　Sub Nmc_ExpMode(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal EM6_data As Long, ByVal EM7_data As Long) **VB.NET**　Sub Nmc_ExpMode(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal EM6_data As Integer, ByVal EM7_data As Integer) **C#.NET**　void MC8000P.Nmc_ExpMode(int No, int IcNo, AXIS Axis, int EM6_data, int EM7_data); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 Axis　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 EM6_data　拡張モードレジスタ EM6 に設定するデータ EM7_data　拡張モードレジスタ EM7 に設定するデータ **戻り値** なし **使用例**　X軸拡張モードに、全フィルタ有効, 遅延512μs, 自動原点出しｽﾃｯﾌﾟ1,2,4の実行を設定する [VC] Nmc_ExpMode(No, IcNo, AXIS_X, 0x5F00, 0x0045); [VB] Call Nmc_ExpMode(No, IcNo, AXIS_X, &H5F00, &H0045) [C#] MC8000P.Nmc_ExpMode(No, IcNo, AXIS.X, 0x5F00, 0x0045);

関数名	機能　及び　内容
Nmc_SyncMode	同期動作モードを設定する。　※MCX314As専用 **VC**　　void Nmc_SyncMode(int No, int IcNo, int Axis, long SM6_data, long SM7_data); **VB**　　Sub Nmc_SyncMode(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal SM6_data As Long, ByVal SM7_data As Long) **VB.NET**　Sub Nmc_SyncMode(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal SM6_data As Integer, ByVal SM7_data As Integer) **C#.NET**　void MC8000P.Nmc_SyncMode(int No, int IcNo, AXIS Axis, int SM6_data, int SM7_data); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 Axis　　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 SM6_data　同期動作モードレジスタ SM6 に設定するデータ SM7_data　同期動作モードレジスタ SM7 に設定するデータ **戻り値** なし **使用例**　「Ｘ軸ドライブ終了時にＹ軸＋方向定量ドライブを開始する」を設定する。 ① Ｘ軸同期動作モードに、起動要因 D-END でＹ軸起動を設定する ② Ｙ軸同期動作モードに、動作として＋方向定量ドライブ(FDRV+) を設定する [VC]　① Nmc_SyncMode(No, IcNo, AXIS_X, 0x2020, 0); ② Nmc_SyncMode(No, IcNo, AXIS_Y, 0, 0x0001); [VB]　① Call Nmc_SyncMode(No, IcNo, AXIS_X, &H2020, 0) ② Call Nmc_SyncMode(No, IcNo, AXIS_Y, 0, &H0001) [C#]　① MC8000P.Nmc_SyncMode(No, IcNo, AXIS.X, 0x2020, 0); ② MC8000P.Nmc_SyncMode(No, IcNo, AXIS.Y, 0, 0x0001);
Nmc_HomeMode	自動原点出しモードを設定する。　※MCX304専用関数 **VC**　　void Nmc_HomeMode(int No, int IcNo, int Axis, long WR6_data); **VB**　　Sub Nmc_HomeMode(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal WR6_data As Long) **VB.NET**　Sub Nmc_HomeMode(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal WR6_data As Integer) **C#.NET** void MC8000P.Nmc_HomeMode(int No, int IcNo, AXIS Axis, int WR6_data); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 Axis　　データを設定する軸。複数軸指定可能。詳細は補足説明(2)参照。 WR6_data　WR6 に設定するデータ **戻り値** なし **使用例**　　Ｘ軸自動原点出しモードに、自動原点出しｽﾃｯﾌﾟ1,2,4の実行を設定する [VC]　Nmc_HomeMode(No, IcNo, AXIS_X, 0x0045); [VB]　Call Nmc_HomeMode(No, IcNo, AXIS_X, &H0045) [C#]　MC8000P.Nmc_HomeMode(No, IcNo, AXIS.X, 0x0045);

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

関数名	機能　及び　内容
Nmc_ReadAccDec	現在加／減速度を読み出す。 ドライブ中の現在加速度、または減速度の値を読み出す。 ドライブ停止時の読み出しデータは不定です。 **VC**　long　Nmc_ReadAccDec(int No, int IcNo, int Axis); **VB**　Function Nmc_ReadAccDec(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadAccDec(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　MC8000P.Nmc_ReadAccDec(int No, int IcNo, AXIS Axis); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを読み出す軸。Ｘ軸はAXIS_X, Ｙ軸はAXIS_Y, Ｚ軸はAXIS_Z, Ｕ軸はAXIS_Uを指定する。 詳細は補足説明(2)参照。 **戻り値** 現在加／減速度 **使用例** [VC]　Data = Nmc_ReadAccDec(No, IcNo, AXIS_U);　// Ｕ軸の現在加／減速度を読み出す [VB]　Data = Nmc_ReadAccDec(No, IcNo, AXIS_U) [C#]　Data = MC8000P.Nmc_ReadAccDec(No, IcNo, AXIS.U);
Nmc_ReadSyncBuff	同期バッファレジスタを読み出す。　※MCX314As専用 **VC**　long　Nmc_ReadSyncBuff(int No, int IcNo, int Axis); **VB**　Function Nmc_ReadSyncBuff(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long) As Long **VB.NET**　Function Nmc_ReadSyncBuff(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer **C#.NET**　int　MC8000P.Nmc_ReadSyncBuff(int No, int IcNo, AXIS Axis); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを読み出す軸。Ｘ軸はAXIS_X, Ｙ軸はAXIS_Y, Ｚ軸はAXIS_Z, Ｕ軸はAXIS_Uを指定する。 詳細は補足説明(2)参照。 **戻り値** 同期バッファレジスタの値 **使用例** [VC]　Data = Nmc_ReadSyncBuff(No, IcNo, AXIS_X);　// Ｘ軸の同期バッファレジスタを読み出す [VB]　Data = Nmc_ReadSyncBuff(No, IcNo, AXIS_X) [C#]　Data = MC8000P.Nmc_ReadSyncBuff(No, IcNo, AXIS.X);

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_GetDriveStatus</td><td>
ドライブ状態を取得する。指定軸のドライブが終了したか調べる時に使用する。

VC　　int　　Nmc_GetDriveStatus(int No, int IcNo, int Axis);
VB　　Function Nmc_GetDriveStatus(ByVal No As Long, ByVal IcNo As Long,ByVal Axis As Long) As LongVB.NET　Function Nmc_GetDriveStatus(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer) As Integer
C#.NET　int　　MC8000P.Nmc_ReadSyncBuff(int No, int IcNo, AXIS Axis);

入力パラメータ
　No　　ボード番号（ボード上のロータリースイッチの値(0～15)）
　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。
　Axis　ドライブ状態を取得する軸。複数軸指定可能。
　　詳細は補足説明(2)参照。

戻り値
　指定した全ての軸のドライブが終了している場合は０を返す。
　指定した軸のうち１つ以上がドライブ中の場合は、０以外を返す。

使用例
　[VC]　if(Nmc_GetDriveStatus(No, IcNo, AXIS_X) == 0)　　// Ｘ軸がドライブ終了の場合
　　AfxMessageBox("Ｘ軸ドライブ終了");
　　else
　　AfxMessageBox("Ｘ軸ドライブ中");
　[VB]　If Nmc_GetDriveStatus(No, IcNo, AXIS_X) = 0 Then　　' Ｘ軸がドライブ終了の場合
　　Call MsgBox("Ｘ軸ドライブ終了")
　　Else
　　Call MsgBox("Ｘ軸ドライブ中")
　　End If
　[C#]　if(MC8000P.Nmc_GetDriveStatus(No, IcNo, AXIS.X) == 0)　// Ｘ軸がドライブ終了の場合
　　MessageBox.Show("Ｘ軸ドライブ終了");
　　else
　　MessageBox.Show("Ｘ軸ドライブ中");
</td></tr>
<tr><td>Nmc_GetCNextStatus</td><td>
連続補間次データ書込み可能状態を取得する。　※MCX314As専用
連続補間実行中に次データ書き込み可能になったか調べる時に使用する。

VC　　int　　Nmc_GetCNextStatus(int No, int IcNo);
VB　　Function Nmc_GetCNextStatus(ByVal No As Long, ByVal IcNo As Long) As Long
VB.NET　Function Nmc_GetCNextStatus(ByVal No As Integer, ByVal IcNo As Integer) As Integer
C#.NET　int　　MC8000P.Nmc_GetCNextStatus(int No, int IcNo);

入力パラメータ
　No　　ボード番号（ボード上のロータリースイッチの値(0～15)）
　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。

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

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

関数名	機能　及び　内容
Nmc_GetBpSc	ＢＰ補間スタックカウンタの値を取得する。　※MCX314As専用 **VC**　　int　　Nmc_GetBpSc(int No, int IcNo); **VB**　　Function Nmc_GetBpSc(ByVal No As Long, ByVal IcNo As Long) As Long **VB.NET**　Function Nmc_GetBpSc(ByVal No As Integer, ByVal IcNo As Integer) As Integer **C#.NET**　int　　MC8000P.Nmc_GetBpSc(int No, int IcNo); **入力パラメータ** 　No　ボード番号（ボード上のロータリースイッチの値(0～15)） 　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 **戻り値** 　現在のビットパターン補間スタックカウンタの値 **使用例** 　[VC]　Data = Nmc_GetBpSc(No, IcNo);　　// ＢＰ補間スタックカウンタの値を取得 　[VB]　Data = Nmc_GetBpSc(No, IcNo) 　[C#]　Data = MC8000P.Nmc_GetBpSc(No, IcNo);
Nmc_WriteRegSetAxis	指定軸の指定したライトレジスタ(ＷＲ１～ＷＲ３のいずれか)にデータを書き込む。 **VC**　　void Nmc_WriteRegSetAxis(int No, int IcNo, int Axis, int RegNumber, long wdata); **VB**　　Sub　Nmc_WriteRegSetAxis(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, 　　ByVal RegNumber As Long, ByVal wdata As Long) **VB.NET**　Sub　Nmc_WriteRegSetAxis(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, 　　ByVal RegNumber As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_WriteRegSetAxis(int No, int IcNo, AXIS Axis, int RegNumber, int wdata); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～15)） 　IcNo　　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 　Axis　　データを書き込む軸。複数軸指定可能。詳細は補足説明(2)参照。 　RegNumber　データを書き込むライトレジスタの番号 　　[VC][VB]　RR1はMCX_RR1, RR2はMCX_RR2 を指定する。 　　[C#]　　RR1はREG_MCX.RR1, RR2はREG_MCX.RR2 を指定する。補足説明(1)参照。 　wdata　　書き込むデータ **戻り値** 　なし **使用例**　全軸のＷＲ２にALARM有効(2000)Hを書き込む 　[VC]　Nmc_WriteRegSetAxis(No, IcNo, AXIS_ALL, MCX_WR2, 0x2000); 　[VB]　Call Nmc_WriteRegSetAxis(No, IcNo, AXIS_ALL, MCX_WR2, &H2000) 　[C#]　MC8000P.Nmc_WriteRegSetAxis(No, IcNo, AXIS.ALL, REG_MCX.WR2, 0x2000);

関数名	機能　及び　内容
Nmc_ReadRegSetAxis	指定軸の指定したリードレジスタ(ＲＲ１、ＲＲ２のどちらか)のデータを読み出す。 **VC**　　long　　Nmc_ReadRegSetAxis(int No, int IcNo, int Axis, int RegNumber); **VB**　　Function Nmc_ReadRegSetAxis(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal RegNumber As Long) As Long **VB.NET**　Function Nmc_ReadRegSetAxis(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal RegNumber As Integer) As Integer **C#.NET**　int　　MC8000P.Nmc_ReadRegSetAxis(int No, int IcNo, AXIS Axis, int RegNumber); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 Axis　　データを読み出す軸。 Ｘ軸はAXIS_X, Ｙ軸はAXIS_Y, Ｚ軸はAXIS_Z, Ｕ軸はAXIS_Uを指定する。 詳細は補足説明(2)参照。 RegNumber　データを読み出すリードレジスタの番号 [VC][VB]　RR1はMCX_RR1, RR2はMCX_RR2 を指定する。 [C#]　　RR1はREG_MCX.RR1, RR2はREG_MCX.RR2 を指定する。補足説明(1)参照。 **戻り値** 指定軸の指定したリードレジスタのデータ **使用例**　　Ｘ軸のＲＲ１のデータを読み出す [VC]　Data = Nmc_ReadRegSetAxis(No, IcNo, AXIS_X, MCX_RR1); [VB]　Data = Nmc_ReadRegSetAxis(No, IcNo, AXIS_X, MCX_RR1) [C#]　Data = MC8000P.Nmc_ReadRegSetAxis(No, IcNo, AXIS.X, REG_MCX.RR1);
Nmc_WriteData	指定軸に指定したパラメータのデータを書き込む。(データ書き込み命令を実行する) **VC**　　void Nmc_WriteData(int No, int IcNo, int Axis, int cmd, long wdata); **VB**　　Sub　Nmc_WriteData(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal cmd As Long, ByVal wdata As Long) **VB.NET**　Sub　Nmc_WriteData(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal cmd As Integer, ByVal wdata As Integer) **C#.NET**　void MC8000P.Nmc_WriteData(int No, int IcNo, AXIS Axis, CMD cmd, int wdata); **入力パラメータ** No　　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 Axis　データを書き込む軸。複数軸指定可能。詳細は補足説明(2)参照。 cmd　データ書き込み命令コード((00)H～(0E)H, (61)H)。例：レンジ設定は(00)H。 ※MCX304は(08)H,(0E)Hを除く。詳細は補足説明(1)参照。 wdata　書き込むデータ **戻り値** なし **使用例**　　全軸のドライブ速度に1000を設定する。ドライブ速度の命令コードは(05)H [VC]　Nmc_WriteData(No, IcNo, AXIS_ALL, 0x05, 1000); [VB]　Call Nmc_WriteData(No, IcNo, AXIS_ALL, &H05, 1000) [C#]　MC8000P.Nmc_WriteData(No, IcNo, AXIS.ALL, CMD.CMD_Speed, 1000);

関数名	機能　及び　内容
Nmc_WriteData2	指定軸に拡張モード、あるいは同期動作モードのデータを書き込む。　　※MCX314As専用 (データ書き込み命令を実行する) **VC**　　void Nmc_WriteData2(int No, int IcNo, int Axis, int cmd, long WR6_data, long WR7_data); **VB**　　Sub Nmc_WriteData2(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal cmd As Long, ByVal WR6_data As Long, ByVal WR7_data As Long) **VB.NET**　Sub Nmc_WriteData2(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal cmd As Integer, ByVal WR6_data As Integer, ByVal WR7_data As Integer) **C#.NET**　void MC8000P.Nmc_WriteData2(int No, int IcNo, AXIS Axis, CMD cmd, int WR6_data, int WR7_data); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～15)） 　IcNo　　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 　Axis　　データを書き込む軸。複数軸指定可能。詳細は補足説明(2)参照。 　cmd　　データ書き込み命令コード。拡張モードは(60)H, 同期動作モードは(64)Hを指定する。 　WR6_data　拡張モードはEM6に, 同期動作モードはSM6に書き込むデータ 　WR7_data　拡張モードはEM7に, 同期動作モードはSM7に書き込むデータ **戻り値** 　なし **使用例**　　Ｘ軸の拡張モードに、EM6データ(5F00)H, EM7データ(45)Hを書き込む。 　[VC]　Nmc_WriteData2(No, IcNo, AXIS_X, 0x60, 0x5F00, 0x0045); 　[VB]　Call Nmc_WriteData2(No, IcNo, AXIS_X, &H60, &H5F00, &H45) 　[C#]　MC8000P.Nmc_WriteData2(No, IcNo, AXIS.X, CMD.CMD_ExpMode, 0x5F00, 0x0045);
Nmc_ReadData	データ読み出し命令を実行し、データを読み出す。 **VC**　　long　　Nmc_ReadData(int No, int IcNo, int Axis, int cmd); **VB**　　Function Nmc_ReadData(ByVal No As Long, ByVal IcNo As Long, ByVal Axis As Long, ByVal cmd As Long) As Long **VB.NET**　Function Nmc_ReadData(ByVal No As Integer, ByVal IcNo As Integer, ByVal Axis As Integer, ByVal cmd As Integer) As Integer **C#.NET**　int　　MC8000P.Nmc_ReadData(int No, int IcNo, AXIS Axis, CMD cmd); **入力パラメータ** 　No　　ボード番号（ボード上のロータリースイッチの値(0～15)） 　IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。 　Axis　データを読み出す軸。Ｘ軸はAXIS_X, Ｙ軸はAXIS_Y, Ｚ軸はAXIS_Z, Ｕ軸はAXIS_Uを指定する。 　cmd　データ読み出し命令コード((10)H～(14)H)。例：論理位置カウンタ読み出しは(10)H。 　　※MCX304は(14)Hを除く。 **戻り値** 　読み出したデータ **使用例** 　[VC]　Data = Nmc_ReadData(No, IcNo, AXIS_X, 0x10);　// Ｘ軸の論理位置カウンタを読み出す。 　[VB]　Data = Nmc_ReadData(No, IcNo, AXIS_X, &H10) 　[C#]　Data = MC8000P.Nmc_ReadData(No, IcNo, AXIS.X, CMD.CMD_ReadLp);

関数名	機能　及び　内容
Nmc_2BPExec	指定した補間データで２軸ビットパターン補間を実行する。　※MCX314As専用 この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。 **VC**　DWORD　Nmc_2BPExec(int No, int IcNo, DATA_2BP* pData2Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_2BPExec(ByVal No As Long, ByVal IcNo As Long, ByRef pData2Bp As DATA_2BP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_2BPExec(ByVal No As Integer, ByVal IcNo As Integer, ByRef pData2Bp As DATA_2BP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8000P.Nmc_2BPExec(int No, int IcNo, DATA_2BP[] pData2Bp, int DataCnt, int IpAxis, bool ContinueFlg); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0,2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 pData2Bp　２軸ＢＰ補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2BPのｱﾄﾞﾚｽ)。DATA_2BPに実行する補間データをセットし、アドレスを指定する。DATA_2BPについては補足説明(3)参照。 DataCnt　２軸ＢＰ補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 ContinueFlg　BP補間が途中で停止した（速度が速く次データのスタックが間に合わなかった）場合に続けるかどうかを設定する。 [VC]　TRUE　:続ける、FALSE　:続けない。省略可能。省略時FALSE。 [VB]　True　:続ける、False　:続けない [C#]　true　:続ける、false　:続けない **戻り値** 補間処理が正常終了した場合は BP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード BP_CNT_ERR　指定したデータ数が範囲外です BP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です BP_PARAM_ERR　引数の値が正しくない BP_NOT_OPEN_ERR　指定したボードがオープンされていない BP_OTHER_ERR　その他のエラー BP_STOP　BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった) BP_USER_STOP　BP補間実行中にユーザーが中断した BP_DRIVE_ERR　BP補間実行中にボードでエラーが発生した（RR0にエラー情報がセットされた） **使用例** [VC] // ２軸ＢＰ補間データ　BP1P,　BP1M,　BP2P,　BP2M DATA_2BP Data2Bp[2] = {{0x0000, 0x2BFF, 0xFFD4, 0x0000}, {0xF6FE, 0x0000, 0x000F, 0x3FC0}}; Nmc_WriteReg5(No, IcNo, 0x04);　// 補間軸設定。主軸：Ｘ、第２軸：Ｙ Ret = Nmc_2BPExec(No, IcNo, Data2Bp, 2, 0x04);　// ２軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y軸 if(Ret == BP_END)　AfxMessageBox("正常終了");　// 戻り値正常 [VB] Dim Data2Bp(1) As DATA_2BP　'２軸ＢＰ補間データ '２軸ＢＰ補間データ設定 Data2Bp(0).Bp1p = &H0:　Data2Bp(0).Bp1m = &H2BFF Data2Bp(0).Bp2p = &HFFD4:　Data2Bp(0).Bp2m = &H0 Data2Bp(1).Bp1p = &HF6FE:　Data2Bp(1).Bp1m = &H0 Data2Bp(1).Bp2p = &HF:　Data2Bp(1).Bp2m = &H3FC0

```
        Call Nmc_WriteReg5(No, IcNo, &H4)                    '補間軸設定。主軸：Ｘ、第２軸：Ｙ
        Ret = Nmc_2BPExec(No, IcNo, Data2Bp(0), 2, &H4, False) '２軸ＢＰ補間実行。ﾃﾞｰﾀ数2,
X,Y軸
        If Ret = BP_END Then                                  '戻り値正常
            Call MsgBox("正常終了")
        End If

    [C#]
        DATA_2BP [] Data2Bp = new DATA_2BP[4];              // ２軸ＢＰ補間データ
        // 補間データ設定
        Data2Bp[0].Bp1p = 0xFF30;// 1111 1111 0011 0000     BP1＋方向     10パルス
        Data2Bp[0].Bp1m = 0;            // 0000 0000 0000 0000     BP1－方向     0パルス
        Data2Bp[0].Bp2p = 0;            // 0000 0000 0000 0000     BP2＋方向     0パルス
        Data2Bp[0].Bp2m = 0x84FF;// 1000 0100 1111 1111     BP2－方向     10パルス

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

        MC8000P.Nmc_WriteReg5(No, IcNo, IpAxis);                  // 補間軸設定
        MC8000P.Nmc_StartSpd(No, IcNo, AXIS.ALL, 1);        // 初速度設定
        MC8000P.Nmc_Speed(gBoardNo, IcNo, AXIS.ALL, 1);           // ドライブ速度設定
        // ２軸ＢＰ補間実行
        Ret = MC8000P.Nmc_2BPExec(No, IcNo, Data2Bp, DataCnt, Axis, ContinueFlg);
        if(Ret == Nmc_Status.BP_END)                        // 戻り値正常
```

関数名	機能　及び　内容

Nmc_3BPExec	指定した補間データで３軸ビットパターン補間を実行する。　※MCX314As専用
	この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。
	
	VC　DWORD　Nmc_3BPExec(int No, int IcNo, DATA_3BP* pData3Bp, int DataCnt, int IpAxis,
	BOOL ContinueFlg = FALSE);
	
	VB　Function Nmc_3BPExec(ByVal No As Long, ByVal IcNo As Long, ByRef pData3Bp As DATA_3BP,
	ByVal DataCnt As Long, ByVal IpAxis As Long,
	ByVal ContinueFlg As Long) As Long
	
	VB.NET　Function Nmc_3BPExec(ByVal No As Integer, ByVal IcNo As Integer, ByRef pData3Bp As DATA_3BP,
	ByVal DataCnt As Integer, ByVal IpAxis As Integer,
	ByVal ContinueFlg As Integer) As Integer
	C#.NET　Nmc_Status MC8000P.Nmc_3BPExec(int No, int IcNo, DATA_3BP[] pData3Bp, int DataCnt,
	int IpAxis, bool ContinueFlg);
	
	入力パラメータ
	No　ボード番号（ボード上のロータリースイッチの値(0～15)）
	IcNo　IC番号(0～1)。搭載ICが1つの時は0,2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照
	pData3Bp　３軸ＢＰ補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_3BPのｱﾄﾞﾚｽ)。
	DATA_3BPに実行する補間データをセットし、アドレスを指定する。
	DATA_3BPについては補足説明(3)参照。
	DataCnt　３軸ＢＰ補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。
	IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。
	ContinueFlg BP補間が途中で停止した（速度が速く次データのスタックが間に合わなかった）場合に続けるかどうかを設定する。
	[VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。
	[VB]　True　：続ける、False　：続けない
	[C#]　true　：続ける、false　：続けない
	
	戻り値
	補間処理が正常終了した場合は BP_END が返ります。
	補間処理でエラーが発生した場合は、下記エラーコードが返ります。
	[C#]の場合は補足説明(1)参照。
	■正常終了
	BP_END　BP補間処理正常終了
	
	■エラーコード
	BP_CNT_ERR　指定したデータ数が範囲外です
	BP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です
	BP_PARAM_ERR　引数の値が正しくない
	BP_NOT_OPEN_ERR　指定したボードがオープンされていない
	BP_OTHER_ERR　その他のエラー
	BP_STOP　BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった)
	BP_USER_STOP　BP補間実行中にユーザーが中断した
	BP_DRIVE_ERR　BP補間実行中にボードでエラーが発生した（RR0にエラー情報がセットされた）
	
	使用例
	[VC] // ３軸ＢＰ補間データ　BP1P,　BP1M,　BP2P,　BP2M,　BP3P,　BP3M
	DATA_3BP Data3Bp[2] = {{0xFF30,　0,　0, 0x84FF,　0, 0xAC35},
	{0xAC35,　0,　0xC000, 0x36E7, 0xC000, 0x3F3F}};
	
	Nmc_WriteReg5(No, IcNo, 0x24);　// 補間軸設定。主軸：X, 第2軸：Y, 第3軸：Z
	Ret = Nmc_3BPExec(No, IcNo, Data3Bp, 2, 0x24); // ３軸ＢＰ補間実行。ﾃﾞｰﾀ数2, X,Y,Z軸
	if(Ret == BP_END)　AfxMessageBox("正常終了"); // 戻り値正常
	
	[VB] Dim Data3Bp(1) As DATA_3BP　'３軸ＢＰ補間データ
	'３軸ＢＰ補間データ設定
	Data3Bp(0).Bp1p = &HFF30: Data3Bp(0).Bp1m = &H0
	Data3Bp(0).Bp2p = &H0:　Data3Bp(0).Bp2m = &H84FF
	Data3Bp(0).Bp3p = &H0:　Data3Bp(0).Bp3m = &HAC35
	Data3Bp(1).Bp1p = &HAC35: Data3Bp(1).Bp1m = &H0

```
        Data3Bp(1).Bp2p = &HC000: Data3Bp(1).Bp2m = &H36E7
        Data3Bp(1).Bp3p = &HC000: Data3Bp(1).Bp3m = &H3F3F

        Call Nmc_WriteReg5(No, IcNo, &H24)                    '補間軸設定。主軸：X, 第2軸：Y, 第
3軸：Z
        Ret = Nmc_3BPExec(No, IcNo, Data3Bp(0), 2, &H24, False) '３軸ＢＰ補間実行。ﾃﾞｰﾀ数2,
X,Y,Z軸
        If Ret = BP_END Then                                  '戻り値正常
            Call MsgBox("正常終了")
        End If

    [C#]
        DATA_3BP [] Data3Bp = new DATA_3BP[4];                // ３軸ＢＰ補間データ
        // 補間データ設定
        Data3Bp[0].Bp1p = 0xFF30;// 1111 1111 0011 0000      BP1＋方向      10パルス
        Data3Bp[0].Bp1m = 0;             // 0000 0000 0000 0000      BP1－方向       0パルス
        Data3Bp[0].Bp2p = 0;             // 0000 0000 0000 0000      BP2＋方向       0パルス
        Data3Bp[0].Bp2m = 0x84FF;// 1000 0100 1111 1111      BP2－方向      10パルス
        Data3Bp[0].Bp3p = 0;             // 0000 0000 0000 0000      BP3＋方向       0パルス
        Data3Bp[0].Bp3m = 0xAC35;// 1010 1100 0011 0101      BP3－方向       8パルス

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

        MC8000P.Nmc_WriteReg5(No, IcNo, IpAxis);                    // 補間軸設定
        MC8000P.Nmc_StartSpd(No, IcNo, AXIS.ALL, 1);          // 初速度設定
        MC8000P.Nmc_Speed(No, IcNo, AXIS.ALL, 1);             // ドライブ速度設定
        // ３軸ＢＰ補間実行
        Ret = MC8000P.Nmc_3BPExec(No, IcNo, Data3Bp, DataCnt, IpAxis, ContinueFlg);
        if(Ret == Nmc_Status.BP_END)                          // 戻り値正常
```

関数名	機能　及び　内容
Nmc_2BPExec_BG	指定した補間データで２軸ビットパターン補間をバックグラウンドで実行する。　※MCX314As専用 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_BP_ENDメッセージを送信し、終了ステータスを渡します。 **VC**　DWORD　Nmc_2BPExec_BG(HWND User_hWnd, int No, int IcNo, DATA_2BP* pData2Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_2BPExec_BG(ByVal User_hWnd As Long, ByVal No As Long, ByVal IcNo As Long, ByRef pData2Bp As DATA_2BP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_2BPExec_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByRef pData2Bp As DATA_2BP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8000P.Nmc_2BPExec_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_2BP[] pData2Bp, int DataCnt, int IpAxis, bool ContinueFlg); **入力パラメータ** User_hWnd　ユーザーアプリケーションのウィンドウハンドル No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0,2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 pData2Bp　２軸ＢＰ補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2BPのｱﾄﾞﾚｽ)。 DATA_2BPに実行する補間データをセットし、アドレスを指定する。 DATA_2BPについては補足説明(3)参照。 DataCnt　２軸ＢＰ補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 ContinueFlg BP補間が途中で停止した（速度が速く次データのスタックが間に合わなかった）場合に続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** バックグラウンドで補間処理が正常に開始した場合は BP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常開始 BP_START　バックグラウンドでBP補間処理が正常に開始した ■エラーコード(補間開始前のエラー) BP_CNT_ERR　指定したデータ数が範囲外です BP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です BP_THREAD_ERR　スレッドを起動できませんでした BP_MALLOC_ERR　メモリを確保できませんでした BP_PARAM_ERR　引数の値が正しくない BP_NOT_OPEN_ERR　指定したボードがオープンされていない BP_OTHER_ERR　その他のエラー バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、WM_BP_ENDメッセージを送信します。WM_BP_ENDのメッセージ受信関数で受け取る第１引数にボード番号を第２引数に終了ステータスを渡します。 その終了ステータスは、補間が正常終了した場合は BP_END です。 補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード(補間開始後のエラー) BP_STOP　BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった) BP_USER_STOP　BP補間実行中にユーザーが中断した BP_DRIVE_ERR　BP補間実行中にボードでエラーが発生した（RR0にエラー情報がセットされた）

使用例

[VC]

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

        Nmc_WriteReg5(No, IcNo, 0x04);                      // 補間軸設定。主軸：Ｘ、第２軸：
Ｙ
        Ret = Nmc_2BPExec_BG(hWnd, No, IcNo, Data2Bp, 2, 0x04);// 2軸ＢＰ補間実行。ﾃﾞｰﾀ数2,
X,Y軸
        if(Ret == BP_START)   AfxMessageBox("補間開始");  // 戻り値正常(補間開始)
    }

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

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

[VB]

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

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

      Call Nmc_WriteReg5(No, IcNo, &H4)                         '補間軸設定。主軸:X, 第2軸:Y
      Ret = Nmc_2BPExec_BG(hWnd,No,IcNo, Data2Bp(0), 2, &H4, False) '2軸BP補間実行。ﾃﾞｰﾀ数2,
X,Y軸
      If Ret = BP_START Then                                    '戻り値正常(補間開始)
          Call MsgBox("補間開始")
      End If
    End Sub
```

ＶＢの場合は次のメッセージ受信関数

```
    'WM_BP_ENDメッセージ受信関数
    Function WindowProc(ByVal hw As Long, ByVal uMsg As Long,
                        ByVal wParam As Long, ByVal lParam As Long) As Long
        If uMsg = WM_BP_END Then          'ＢＰ補間終了メッセージ
            If lParam = BP_END Then       '戻り値正常(補間終了)
                Call MsgBox("補間正常終了")
            End If
        End If

        WindowProc = CallWindowProc(glpPrevWndProc, hw, uMsg, wParam, lParam)

    End Function
```

ＶＢ．ＮＥＴの場合は次のメッセージ受信関数

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

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

```
     MyBase.WndProc(m)

 End Sub


[C#]
    DATA_2BP [] Data2Bp = new DATA_2BP[4];                  // ２軸ＢＰ補間データ
    // 補間データ設定
    Data2Bp[0].Bp1p = 0xFF30;// 1111 1111 0011 0000      BP1＋方向      10パルス
    Data2Bp[0].Bp1m = 0;             // 0000 0000 0000 0000      BP1－方向       0パルス
    Data2Bp[0].Bp2p = 0;             // 0000 0000 0000 0000      BP2＋方向       0パルス
    Data2Bp[0].Bp2m = 0x84FF;// 1000 0100 1111 1111      BP2－方向      10パルス

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

    MC8000P.Nmc_WriteReg5(No, IcNo, IpAxis);                        // 補間軸設定
    MC8000P.Nmc_StartSpd(No, IcNo, AXIS.ALL, 1);            // 初速度設定
    MC8000P.Nmc_Speed(gBoardNo, IcNo, AXIS.ALL, 1);                 // ドライブ速度設定
    // ２軸ＢＰ補間バックグラウンド実行
    Ret = MC8000P.Nmc_2BPExec_BG((System.IntPtr)parent.Handle, No, IcNo, Data2Bp,
                                DataCnt, IpAxis, ContinueFlg);
    if(Ret == Nmc_Status.BP_START)                          // 補間開始正常の場合

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

      }
    }
```

関数名	機能　及び　内容
Nmc_3BPExec_BG	指定した補間データで３軸ビットパターン補間をバックグラウンドで実行する。　※MCX314As専用 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_BP_ENDメッセージを送信し、終了ステータスを渡します。 **VC**　DWORD　Nmc_3BPExec_BG(HWND User_hWnd, int No, int IcNo, DATA_3BP* pData3Bp, int DataCnt, int IpAxis, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_3BPExec_BG(ByVal User_hWnd As Long, ByVal No As Long, ByVal IcNo As Long, ByRef pData3Bp As DATA_3BP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_3BPExec_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByRef pData3Bp As DATA_3BP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8000P.Nmc_3BPExec_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_3BP[] pData3Bp, int DataCnt, int IpAxis, bool ContinueFlg); **入力パラメータ** User_hWnd　ユーザーアプリケーションのウィンドウハンドル No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0,2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 pData3Bp　３軸ＢＰ補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_3BPのｱﾄﾞﾚｽ)。 DATA_3BPに実行する補間データをセットし、アドレスを指定する。 DATA_3BPについては補足説明(3)参照。 DataCnt　３軸ＢＰ補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 ContinueFlg BP補間が途中で停止した（速度が速く次データのスタックが間に合わなかった）場合に続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** バックグラウンドで補間処理が正常に開始した場合は BP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常開始 BP_START　バックグラウンドでBP補間処理が正常に開始した ■エラーコード(補間開始前のエラー) BP_CNT_ERR　指定したデータ数が範囲外です BP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です BP_THREAD_ERR　スレッドを起動できませんでした BP_MALLOC_ERR　メモリを確保できませんでした BP_PARAM_ERR　引数の値が正しくない BP_NOT_OPEN_ERR　指定したボードがオープンされていない BP_OTHER_ERR　その他のエラー バックグラウンドで補間処理が正常に開始した後は、補間処理終了時に、指定したウィンドウに対して、WM_BP_ENDメッセージを送信します。WM_BP_ENDのメッセージ受信関数で受け取る第１引数にボード番号を　第２引数に終了ステータスを渡します。 その終了ステータスは、補間が正常終了した場合は BP_END です。 補間実行時にエラーが発生した場合は、下記の補間開始後のエラーコードです。 ■正常終了 BP_END　BP補間処理正常終了 ■エラーコード(補間開始後のエラー) BP_STOP　BP補間が途中で停止した(速度が速く次データのスタックが間に合わなかった) BP_USER_STOP　BP補間実行中にユーザーが中断した BP_DRIVE_ERR　BP補間実行中にボードでエラーが発生した（RR0にエラー情報がセットされた）

使用例

[VC]

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

        Nmc_WriteReg5(No, IcNo, 0x24);                      // 補間軸設定。主軸:X, 第2軸:Y, 第
3軸:Z
        Ret = Nmc_3BPExec_BG(hWnd, No, IcNo, Data3Bp, 2, 0x24);//3軸ＢＰ補間実行。ﾃﾞｰﾀ数2,
X,Y,Z軸
        if(Ret == BP_START)   AfxMessageBox("補間開始");  // 戻り値正常(補間開始)
    }

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

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

[VB]

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

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

      Call Nmc_WriteReg5(No, IcNo, &H24)                    '補間軸設定。主軸:X, 第2軸:Y, 第
3軸:Z
      Ret = Nmc_3BPExec_BG(hWnd,No,IcNo,Data3Bp(0),2,&H24,False)'3軸ＢＰ補間実行。ﾃﾞｰﾀ数2,
X,Y,Z軸
      If Ret = BP_START Then                                '戻り値正常(補間開始)
          Call MsgBox("補間開始")
      End If
    End Sub
```

ＶＢの場合は次のメッセージ受信関数

```
    'WM_BP_ENDメッセージ受信関数
    Function WindowProc(ByVal hw As Long, ByVal uMsg As Long,
                        ByVal wParam As Long, ByVal lParam As Long) As Long
        If uMsg = WM_BP_END Then          'ＢＰ補間終了メッセージ
            If lParam = BP_END Then       '戻り値正常(補間終了)
                Call MsgBox("補間正常終了")
            End If
        End If

        WindowProc = CallWindowProc(glpPrevWndProc, hw, uMsg, wParam, lParam)
    End Function
```

ＶＢ．ＮＥＴの場合は次のメッセージ受信関数

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

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

```
            End If
        End If

        MyBase.WndProc(m)
    End Sub


[C#]
    DATA_3BP [] Data3Bp = new DATA_3BP[4];                  // ３軸ＢＰ補間データ
    // 補間データ設定
    Data3Bp[0].Bp1p = 0xFF30;// 1111 1111 0011 0000      BP1＋方向      10パルス
    Data3Bp[0].Bp1m = 0;             // 0000 0000 0000 0000      BP1－方向       0パルス
    Data3Bp[0].Bp2p = 0;             // 0000 0000 0000 0000      BP2＋方向       0パルス
    Data3Bp[0].Bp2m = 0x84FF;// 1000 0100 1111 1111      BP2－方向      10パルス
    Data3Bp[0].Bp3p = 0;             // 0000 0000 0000 0000      BP3＋方向       0パルス
    Data3Bp[0].Bp3m = 0xAC35;// 1010 1100 0011 0101      BP3－方向       8パルス

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

    MC8000P.Nmc_WriteReg5(No, IcNo, IpAxis);                    // 補間軸設定
    MC8000P.Nmc_StartSpd(No, IcNo, AXIS.ALL, 1);          // 初速度設定
    MC8000P.Nmc_Speed(gBoardNo, IcNo, AXIS.ALL, 1);             // ドライブ速度設定
    // ３軸ＢＰ補間バックグラウンド実行
    Ret = MC8000P.Nmc_3BPExec_BG((System.IntPtr)parent.Handle, No, IcNo, Data3Bp,
                                 DataCnt, IpAxis, ContinueFlg);
    if(Ret == Nmc_Status.BP_START)                          // 補間開始正常の場合

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

      }
    }
```

関数名	機能　及び　内容
Nmc_2CIPExec	指定した補間データで２軸連続補間を実行する。　※MCX314As専用 この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。 **VC**　DWORD　Nmc_2CIPExec(int No, int IcNo, DATA_2CIP* pData2Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_2CIPExec(ByVal No As Long, ByVal IcNo As Long, ByRef pData2Cip As DATA_2CIP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal SpdChgFlg As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_2CIPExec(ByVal No As Integer, ByVal IcNo As Integer, ByRef pData2Cip As DATA_2CIP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer,ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8000P. Nmc_2CIPExec(int No, int IcNo, DATA_2CIP[] pData2Cip, int DataCnt, int IpAxis, bool SpdChgFlg, bool ContinueFlg); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0,2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 pData2Cip　２軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2CIPのｱﾄﾞﾚｽ)。 DATA_2CIPに実行する補間データをセットし、アドレスを指定する。 DATA_2CIPについては補足説明(3)参照。 DataCnt　２軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 SpdChgFlg　補間実行中に速度を変更するかどうかを設定する。変更する場合は補足説明(5)参照。 [VC]　TRUE　：変更する、FALSE　：変更しない。省略可能。省略時FALSE。 [VB]　True　：変更する、False　：変更しない [C#]　true　：変更する、false　：変更しない **変更する選択時**　：DATA_2CIPのSpeedに設定した値を参照します。 Speedに1～8000の値設定・・・設定した速度に変更します。 Speedに0 設定 ・・・・・・・速度は変更しません。 **変更しない選択時**：DATA_2CIPのSpeedに設定した値を参照しません。 ContinueFlg 連続補間が途中で停止した（速度が速く次データのセットが間に合わなかった）場合に続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** 補間処理が正常終了した場合は CIP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常終了 CIP_END　連続補間処理正常終了 ■エラーコード CIP_CNT_ERR　指定したデータ数が範囲外です CIP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です CIP_CMD_ERR　指定したコマンドが間違っています CIP_PARAM_ERR　引数の値が正しくない CIP_NOT_OPEN_ERR　指定したボードがオープンされていない CIP_OTHER_ERR　その他のエラー CIP_STOP　連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった) CIP_USER_STOP　連続補間実行中にユーザーが中断した CIP_DRIVE_ERR　連続補間実行中にボードでエラーが発生した(RR0にエラー情報がセットされた)

使用例

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

         Nmc_WriteReg5(No, IcNo, 0x04);                          // 補間軸設定。主軸：Ｘ、第２
軸：Ｙ
         Ret = Nmc_2CIPExec(No, IcNo, Data2Cip, 2, 0x04);      // ２軸連続補間実行。ﾃﾞｰﾀ数2,
X,Y軸
         if(Ret == CIP_END)   AfxMessageBox("正常終了"); // 戻り値正常

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

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

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

         Call Nmc_WriteReg5(No, IcNo, &H4)                          '補間軸設定。主軸：X、第
２軸：Y
         Ret = Nmc_2CIPExec(No, IcNo, Data2Cip(0), 2, &H4, False, False)'2軸連続補間。ﾃﾞｰﾀ数2,
X,Y軸
         If Ret = CIP_END Then                                      '戻り値正常
             Call MsgBox("正常終了")
         End If

    [C#]
         DATA_2CIP [] Data2Cip = new DATA_2CIP[4];              // ２軸連続補間データ

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

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

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

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

関数名	機能　及び　内容
Nmc_3CIPExec	指定した補間データで３軸連続補間を実行する。　※MCX314As専用 この関数は、補間処理が終了した後に制御を返します。補間が終了するまで制御が戻らないのでアプリケーションでスレッドを作成し、そのスレッドからコールする事を推奨します。 **VC**　DWORD　Nmc_3CIPExec(int No, int IcNo, DATA_3CIP* pData3Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_3CIPExec(ByVal No As Long, ByVal IcNo As Long, ByRef pData3Cip As DATA_3CIP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal SpdChgFlg As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_3CIPExec(ByVal No As Integer, ByVal IcNo As Integer, ByRef pData3Cip As DATA_3CIP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer,ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8000P.Nmc_3CIPExec(int No, IcNo, DATA_3CIP[] pData3Cip, int DataCnt, int IpAxis, bool SpdChgFlg, bool ContinueFlg); **入力パラメータ** No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0,2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 pData3Cip　３軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_3CIPのｱﾄﾞﾚｽ)。 DATA_3CIPに実行する補間データをセットし、アドレスを指定する。 DATA_3CIPについては補足説明(3)参照。 DataCnt　３軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 SpdChgFlg　補間実行中に速度を変更するかどうかを設定する。変更する場合は補足説明(5)参照。 [VC]　TRUE　：変更する、FALSE　：変更しない。省略可能。省略時FALSE。 [VB]　True　：変更する、False　：変更しない [C#]　true　：変更する、false　：変更しない **変更する選択時**　：DATA_3CIPのSpeedに設定した値を参照します。 Speedに1～8000の値設定・・・設定した速度に変更します。 SpeedにO 設定　・・・・・・・速度は変更しません。 **変更しない選択時**：DATA_3CIPのSpeedに設定した値を参照しません。 ContinueFlg 連続補間が途中で停止した（速度が速く次データのセットが間に合わなかった）場合に続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** 補間処理が正常終了した場合は CIP_END が返ります。 補間処理でエラーが発生した場合は、下記エラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常終了 CIP_END　連続補間処理正常終了 ■エラーコード CIP_CNT_ERR　指定したデータ数が範囲外です CIP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です CIP_CMD_ERR　指定したコマンドが間違っています CIP_PARAM_ERR　引数の値が正しくない CIP_NOT_OPEN_ERR　指定したボードがオープンされていない CIP_OTHER_ERR　その他のエラー CIP_STOP　連続補間が途中で停止した(速度が速く次データのセットが間に合わなかった) CIP_USER_STOP　連続補間実行中にユーザーが中断した CIP_DRIVE_ERR　連続補間実行中にボードでエラーが発生した（RR0にエラー情報がセットされた）

使用例

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

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

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

         Nmc_WriteReg5(No, IcNo, 0x24);                    // 補間軸設定。主軸：X, 第2軸：Y, 第
3軸：Z
         Ret = Nmc_3CIPExec(No, IcNo, Data3Cip, 2, 0x24); // ３軸連続補間実行。ﾃﾞｰﾀ数2, X,Y,Z軸
         if(Ret == CIP_END)   AfxMessageBox("正常終了");  // 戻り値正常

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

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

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

         Call Nmc_WriteReg5(No, IcNo, &H24)                  '補間軸設定。主軸:X, 第2軸:Y,
第3軸:Z
         Ret = Nmc_3CIPExec(No,IcNo, Data3Cip(0), 2, &H24, False,False)'3軸連続補間。ﾃﾞｰﾀ数
2,X,Y,Z軸
         If Ret = CIP_END Then                                '戻り値正常
             Call MsgBox("正常終了")
         End If

    [C#]
        DATA_3CIP [] Data3Cip = new DATA_3CIP[4];          // ３軸連続補間データ用
        // 補間データ設定
        Data3Cip[0].EndP1 = 1000;
        Data3Cip[0].EndP2 = 2000;
        Data3Cip[0].EndP3 = 3000;
        Data3Cip[0].Speed = 0;

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

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

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

関数名	機能　及び　内容
Nmc_2CIPExec_BG	指定した補間データで２軸連続補間をバックグラウンドで実行する。　※MCX314As専用 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_CIP_ENDメッセージを送信し、終了ステータスを渡します。 **VC**　DWORD　Nmc_2CIPExec_BG(HWND User_hWnd, int No, int IcNo, DATA_2CIP* pData2Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_2CIPExec_BG(ByVal User_hWnd As Long, ByVal No As Long, ByVal IcNo As Long, ByRef pData2Cip As DATA_2CIP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal SpdChgFlg As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_2CIPExec_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByRef pData2Cip As DATA_2CIP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer,ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8000P.Nmc_2CIPExec_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_2CIP[] pData2Cip, int DataCnt, int IpAxis, bool SpdChgFlg, bool ContinueFlg); **入力パラメータ** User_hWnd　ユーザーアプリケーションのウィンドウハンドル No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0,2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 pData2Cip　２軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_2CIPのｱﾄﾞﾚｽ)。 DATA_2CIPに実行する補間データをセットし、アドレスを指定する。 DATA_2CIPについては補足説明(3)参照。 DataCnt　２軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 SpdChgFlg　補間実行中に速度を変更するかどうかを設定する。変更する場合は補足説明(5)参照。 [VC]　TRUE　：変更する、FALSE　：変更しない。省略可能。省略時FALSE。 [VB]　True　：変更する、False　：変更しない [C#]　true　：変更する、false　：変更しない **変更する選択時**　：DATA_2CIPのSpeedに設定した値を参照します。 Speedに1～8000の値設定・・・設定した速度に変更します。 Speedに0 設定　・・・・・・・速度は変更しません。 **変更しない選択時**：DATA_2CIPのSpeedに設定した値を参照しません。 ContinueFlg 連続補間が途中で停止した（速度が速く次データのセットが間に合わなかった）場合に続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** バックグラウンドで補間処理が正常に開始した場合は CIP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常開始 CIP_START　バックグラウンドで連続補間処理が正常に開始した ■エラーコード(補間開始前のエラー) CIP_CNT_ERR　指定したデータ数が範囲外です CIP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です CIP_THREAD_ERR　スレッドを起動できませんでした CIP_MALLOC_ERR　メモリを確保できませんでした

CIP_CMD_ERR　　指定したコマンドが間違っています
CIP_PARAM_ERR　　引数の値が正しくない
CIP_NOT_OPEN_ERR　指定したボードがオープンされていない
CIP_OTHER_ERR　　その他のエラー

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

■正常終了
CIP_END　　連続補間処理正常終了

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

使用例

[VC]

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

   Nmc_WriteReg5(No, IcNo, 0x04);                          // 補間軸設定。主軸：Ｘ、第２軸：Ｙ
   Ret = Nmc_2CIPExec_BG(hWnd, No, IcNo, Data2Cip, 2, 0x04);// 2軸連続補間実行。ﾃﾞｰﾀ数2, X,Y軸
   if(Ret == CIP_START)   AfxMessageBox("補間開始");      // 戻り値正常(補間開始)
}

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

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

[VB]

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

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

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

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

```
        Ret = Nmc_2CIPExec_BG(hWnd, No, IcNo, Data2Cip(0), 2, &H4, False, False)
        If Ret = CIP_START Then                                    '戻り値正常(補間開始)
            Call MsgBox("補間開始")
        End If
    End Sub
```

ＶＢの場合は次のメッセージ受信関数

```
    'WM_CIP_ENDメッセージ受信関数
    Function WindowProc(ByVal hw As Long, ByVal uMsg As Long,
                        ByVal wParam As Long, ByVal lParam As Long) As Long
        If uMsg = WM_CIP_END Then           '連続補間終了メッセージ
            If lParam = CIP_END Then        '戻り値正常(補間終了)
                Call MsgBox("補間正常終了")
            End If
        End If
        WindowProc = CallWindowProc(glpPrevWndProc, hw, uMsg, wParam, lParam)
    End Function
```

ＶＢ．ＮＥＴの場合は次のメッセージ受信関数

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

```
[C#]
    DATA_2CIP [] Data2Cip = new DATA_2CIP[4];       // ２軸連続補間データ

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

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

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

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

Ｃ＃．ＮＥＴの場合は次のメッセージ受信関数

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

      }
    }
```

関数名	機能　及び　内容
Nmc_3CIPExec_BG	指定した補間データで３軸連続補間をバックグラウンドで実行する。　※MCX314As専用 この関数は、補間処理を開始した直後に制御を返し、バックグラウンドで補間を実行します。 指定したウィンドウに対して、補間終了時にWM_CIP_ENDメッセージを送信し、終了ステータスを渡します。 **VC**　DWORD　Nmc_3CIPExec_BG(HWND User_hWnd, int No, int IcNo, DATA_3CIP* pData3Cip, int DataCnt, int IpAxis, BOOL SpdChgFlg = FALSE, BOOL ContinueFlg = FALSE); **VB**　Function Nmc_3CIPExec_BG(ByVal User_hWnd As Long, ByVal No As Long, ByVal IcNo As Long, ByRef pData3Cip As DATA_3CIP, ByVal DataCnt As Long, ByVal IpAxis As Long, ByVal SpdChgFlg As Long, ByVal ContinueFlg As Long) As Long **VB.NET**　Function Nmc_3CIPExec_BG(ByVal User_hWnd As Integer, ByVal No As Integer, ByVal IcNo As Integer, ByRef pData3Cip As DATA_3CIP, ByVal DataCnt As Integer, ByVal IpAxis As Integer, ByVal SpdChgFlg As Integer,ByVal ContinueFlg As Integer) As Integer **C#.NET**　Nmc_Status MC8000P.Nmc_3CIPExec_BG(System.IntPtr User_hWnd, int No, int IcNo, DATA_3CIP[] pData3Cip, int IpAxis, bool SpdChgFlg, bool ContinueFlg); **入力パラメータ** User_hWnd　ユーザーアプリケーションのウィンドウハンドル No　ボード番号（ボード上のロータリースイッチの値(0～15)） IcNo　IC番号(0～1)。搭載ICが1つの時は0,2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照 pData3Cip　３軸連続補間データの構造体(ユーザー定義型)配列の先頭アドレス(DATA_3CIPのｱﾄﾞﾚｽ)。 DATA_3CIPに実行する補間データをセットし、アドレスを指定する。 DATA_3CIPについては補足説明(3)参照。 DataCnt　３軸連続補間データの数。構造体(ユーザー定義型)配列の配列数を指定する。 IpAxis　補間を実行する軸。WR5のD0～D5(軸指定)の設定値と同じ値を指定する。補足説明(4)参照。 SpdChgFlg　補間実行中に速度を変更するかどうかを設定する。変更する場合は補足説明(5)参照。 [VC]　TRUE　：変更する、FALSE　：変更しない。省略可能。省略時FALSE。 [VB]　True　：変更する、False　：変更しない [C#]　true　：変更する、false　：変更しない **変更する選択時**　：DATA_3CIPのSpeedに設定した値を参照します。 Speedに1～8000の値設定・・・設定した速度に変更します。 Speedに0 設定 ・・・・・・・速度は変更しません。 **変更しない選択時**：DATA_3CIPのSpeedに設定した値を参照しません。 ContinueFlg 連続補間が途中で停止した（速度が速く次データのセットが間に合わなかった）場合に続けるかどうかを設定する。 [VC]　TRUE　：続ける、FALSE　：続けない。省略可能。省略時FALSE。 [VB]　True　：続ける、False　：続けない [C#]　true　：続ける、false　：続けない **戻り値** バックグラウンドで補間処理が正常に開始した場合は CIP_START が返ります。 補間処理が開始する前にエラーが発生した場合は、下記の補間開始前のエラーコードが返ります。 [C#]の場合は補足説明(1)参照。 ■正常開始 CIP_START　バックグラウンドで連続補間処理が正常に開始した ■エラーコード(補間開始前のエラー) CIP_CNT_ERR　指定したデータ数が範囲外です CIP_ALREADY_EXEC　既にＢＰ補間、あるいは連続補間が実行中です CIP_THREAD_ERR　スレッドを起動できませんでした CIP_MALLOC_ERR　メモリを確保できませんでした CIP_CMD_ERR　指定したコマンドが間違っています

CIP_PARAM_ERR　　引数の値が正しくない
CIP_NOT_OPEN_ERR　指定したボードがオープンされていない
CIP_OTHER_ERR　　その他のエラー

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

■正常終了
CIP_END　　連続補間処理正常終了

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

使用例

[VC]

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

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

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

   Nmc_WriteReg5(No, IcNo, 0x24);                    // 補間軸設定。主軸：X, 第2軸：Y, 第3軸：Z
   Ret = Nmc_3CIPExec_BG(hWnd,No, IcNo, Data3Cip, 2, 0x24);//3軸連続補間実行。ﾃﾞｰﾀ数2, X,Y,Z軸
   if(Ret == CIP_START)   AfxMessageBox("補間開始");// 戻り値正常(補間開始)
}

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

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

[VB]

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

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

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

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

第3軸:Z

```
                                                        '3軸連続補間。ﾃﾞｰﾀ数2, X,Y,Z軸
        Ret = Nmc_3CIPExec_BG(hWnd, No, IcNo, Data3Cip(0), 2, &H24, False, False)
        If Ret = CIP_START Then                          '戻り値正常(補間開始)
            Call MsgBox("補間開始")
        End If
    End Sub
```

ＶＢの場合は次のメッセージ受信関数

```
    'WM_CIP_ENDメッセージ受信関数
    Function WindowProc(ByVal hw As Long, ByVal uMsg As Long,
                        ByVal wParam As Long, ByVal lParam As Long) As Long

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

        WindowProc = CallWindowProc(glpPrevWndProc, hw, uMsg, wParam, lParam)

    End Function
```

ＶＢ．ＮＥＴの場合は次のメッセージ受信関数

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

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

        MyBase.WndProc(m)

    End Sub
```

[C#]

```
    DATA_3CIP [] Data3Cip = new DATA_3CIP[4];                   // ３軸連続補間データ用
    // 補間データ設定
    Data3Cip[0].EndP1 = 1000;
    Data3Cip[0].EndP2 = 2000;
    Data3Cip[0].EndP3 = 3000;
    Data3Cip[0].Speed = 0;

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

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

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

Ｃ＃．ＮＥＴの場合は次のメッセージ受信関数

```
    //WM_BP_ENDメッセージ受信関数
    protected override void WndProc(ref Message m)
```

```
        {
          // 元のWndProc呼び出し(コンストラクタの呼び出しを明示的に記述)
          base.WndProc ( ref m );
          if ( m.Msg == (int)MSG_ID.WM_CIP_END )
          {
                if((uint)m.LParam == Nmc_Status.CIP_END)            // BP補間処理正常終了

          }
        }
```

<table>
<tr><th>関数名</th><th>機能　及び　内容</th></tr>
<tr><td>Nmc_IPStop</td><td>補間処理を中断する。　※MCX314As専用
補間ドライブを即停止し、Nmc_xxxの補間関数で実行中の補間処理を終了します。

Nmc_IPStopを使用して補間処理を中断した場合、各補間関数の戻り値は下記のエラーコードです。
◆ＢＰ補間　：　BP_USER_STOP
◆連続補間　：　CIP_USER_STOP

VC　　BOOL　　Nmc_IPStop(int No, int IcNo);
VB　　Function Nmc_IPStop(ByVal No As Long, ByVal IcNo As Long) As Long
VB.NET　Function Nmc_IPStop(ByVal No As Integer, ByVal IcNo As Integer) As Integer
C#.NET　bool　　MC8000P.Nmc_IPStop(int No, int IcNo);
入力パラメータ
No　ボード番号（ボード上のロータリースイッチの値(0～15)）
IcNo　IC番号(0～1)。搭載ICが1つの時は0, 2つ以上の時はIC-Aが0,IC-Bが1。詳細は補足説明(7)参照。

戻り値
[VC]　補間処理の中断に成功するとTRUE、失敗するとFALSE
[VB]　補間処理の中断に成功するとＯ以外、失敗するとＯ
[C#]　補間処理の中断に成功するとtrue、失敗するとfalse
使用例
[VC]　Nmc_IPStop(No, IcNo);　　// 補間処理を中断する
[VB]　Call Nmc_IPStop(No, IcNo)
[C#]　MC8000P.Nmc_IPStop(No, IcNo);</td></tr>
<tr><td>Nmc_IPGetMsgNo</td><td>補間終了時に受信した下記メッセージの引数wParamからボード番号とＩＣ番号を取得する。　※MCX314As専用

◆ＢＰ補間　：　WM_BP_END
◆連続補間　：　WM_CIP_END

VC　　void Nmc_IPGetMsgNo(int wParam, int* No, int* IcNo);
VB　　Sub　Nmc_IPGetMsgNo(ByVal wParam As Long, ByRef No As Long, ByRef IcNo As Long)
VB.NET　Sub　Nmc_IPGetMsgNo(ByVal wParam As Integer, ByRef No As Integer, ByRef IcNo As Integer)
C#.NET　bool　　MC8000P.Nmc_IPStop(int No, int IcNo);
入力パラメータ
wParam　補間終了時に受信したメッセージの引数wParam
No　　[VC]ボード番号を格納する変数のアドレス。
　　[VB]ボード番号を格納する変数。
IcNo　[VC]ＩＣ番号を格納する変数のアドレス。
　　[VB]ＩＣ番号を格納する変数。

戻り値
なし

使用例
[VC]　　int　BdNo;　　// ボード番号
　　int　IcNo;　　// ＩＣ番号
　　Nmc_IPGetMsgNo(wParam, &BdNo, &IcNo);

[VB]　　Dim BdNo As Long　　' ボード番号
　　Dim IcNo As Long　　' ＩＣ番号
　　Call Nmc_IPGetMsgNo(wParam, BdNo, IcNo)

[VB.NET]　Dim BdNo As Integer　' ボード番号
　　Dim IcNo As Integer　' ＩＣ番号
　　Call Nmc_IPGetMsgNo(m.WParam.ToInt32(), BdNo, IcNo)

[VC]　　int　BdNo;　　// ボード番号

[C#]　　int BdNo;　　// ボード番号
　　int IcNo;　　// ＩＣ番号 (0～1)
　　MC8000P.Nmc_IPGetMsgNo(WParam, out BdNo, out IcNo)</td></tr>
</table>

■補足説明

(1) 各定義は、下記のファイルで行っています。

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

ＶＣ、Ｃ＃．ＮＥＴの定義内容を以下に示します。

①レジスタ番号
[VC]

```
   #define MCX_WR0          0x0000 // ＷＲ０
   #define MCX_WR1          0x0001 // ＷＲ１
   #define MCX_WR2          0x0002 // ＷＲ２
   #define MCX_WR3          0x0003 // ＷＲ３
   #define MCX_WR4          0x0004 // ＷＲ４
   #define MCX_WR5          0x0005 // ＷＲ５
   #define MCX_WR6          0x0006 // ＷＲ６
   #define MCX_WR7          0x0007 // ＷＲ７

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

[C#]

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

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

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

    // MSM82C55のアドレス
```

```
WR10   =0x0010,
WR11   =0x0011,
WR12   =0x0012,
// PIX132のアドレス
WR14   =0x0014,
WR15   =0x0015,
WR16   =0x0016,

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

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

// MSM82C55のアドレス
RR10   =0x0010,
RR11   =0x0011,
RR12   =0x0012,
// PIX132のアドレス
RR14   =0x0014,
RR15   =0x0015,
RR16   =0x0016,

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

//■Nmc_ReadReg用
// リードレジスタのアドレス
RR0    =0x0000,
RR1    =0x0001,
RR2    =0x0002,
RR3    =0x0003,
RR4    =0x0004,
RR5    =0x0005,
RR6    =0x0006,
RR7    =0x0007
}
```

（使用例）REG_MCXのWR0の場合は　　REG_MCX.WR0

②軸定義

[VC]

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

[C#]

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

（使用例）全軸の場合は　　AXIS.ALL

③デバイスＩＤ

（注）**MC80**42P,22P及びMC8082Peは、デバイスIDは**MC8082Pと変わらない事に注意してください。**
　　同様にMC8043Peは、デバイスIDは**MC8043Pと変わらない事に注意してください。**

[VC]

```
   #define ID_MC8043P              0xA0A2       // MC8043P及びMC8043Pe
   #define ID_MC8080P              0xA07D  // MC8080P
   #define ID_MC8082P             0xA0D0 // MC8082P,42P,22P及びMC8082Pe
```

（使用例）**MC8082P**,42P,22P及びMC8082Pe**はID_MC8082P、MC8080Pは**ID_MC8080P、**MC8043P**及びMC8082Pe**はID_MC8043P**

[C#]

```
   public enum Dev_ID : ushort
   {
      MC8043P              =0xA0A2,       // MC8043P及びMC8043Pe
      MC8080P              =0xA07D,       // MC8080P
      MC8082P              =0xA0D0              // MC8082P,42P,22P及びMC8082Pe
   }
```

（使用例）**MC8082P**,42P,22P及びMC8082Pe**は**Dev_ID.**MC8082P、MC8080Pは**Dev_ID.**MC8080P**
　　MC8043P及びMC8043Pe**は**Dev_ID.**MC8043P**

④コマンド定義　　※使用できるコマンドは各ＩＣの取扱説明書参照。

[VC]

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

   // 補間命令　　※MCX314As 専用
   #define CMD_IP_2ST              0x30   // ２軸直線補間ドライブ
   #define CMD_IP_3ST              0x31   // ３軸直線補間ドライブ
   #define CMD_IP_CW       0x32   // ＣＷ円弧補間ドライブ
   #define CMD_IP_CCW              0x33   // ＣＣＷ円弧補間ドライブ
   #define CMD_IP_2BP              0x34   // ２軸ビットパターン補間ドライブ
   #define CMD_IP_3BP              0x35   // ３軸ビットパターン補間ドライブ
   #define CMD_BP_ENABLED          0x36   // ＢＰレジスタ書き込み可
   #define CMD_BP_DISABLED         0x37   // ＢＰレジスタ書き込み不可
   #define CMD_BP_STACK            0x38   // ＢＰデータスタック
   #define CMD_BP_CLR              0x39   // ＢＰデータクリア
   #define CMD_IP_1STEP            0x3A   // 補間シングルステップ
   #define CMD_IP_DEC_VALID        0x3B   // 減速有効
```

```
   #define CMD_IP_DEC_INVALID     0x3C   // 減速無効
   #define CMD_IP_INTRPT_CLR      0x3D   // 補間割り込みクリア



   // 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パルス出力
   #define CMD_SYNC_ACTIVE        0x65   // 同期動作起動　※MCX314As 専用
   #define CMD_NOP         0x0F   // ＮＯＰ（軸切り換え用）
```

[C#]

```
public enum CMD : int
{
       // データ書き込み命令
       CMD_Range             = 0x00,     // レンジ設定
       CMD_Jerk                    = 0x01,     // 加速度増加率設定
       CMD_Acc               = 0x02,     // 加速度設定
       CMD_Dec               = 0x03,     // 減速度設定
       CMD_StartSpd          = 0x04,     // 初速度設定
       CMD_Speed             = 0x05,     // ドライブ速度設定
       CMD_Pulse             = 0x06,     // 出力パルス数／補間終点設定
       CMD_DecP                    = 0x07,     // マニュアル減速点設定
       CMD_Center            = 0x08,     // 円弧中心点設定
       CMD_Lp                      = 0x09,     // 論理位置カウンタ設定
       CMD_Ep                      = 0x0A,     // 実位置カウンタ設定
       CMD_CompP             = 0x0B,     // ＣＯＭＰ＋レジスタ設定
       CMD_CompM             = 0x0C,     // ＣＯＭＰ－レジスタ設定
       CMD_AccOfst           = 0x0D,     // 加速カウンタオフセット設定
       CMD_DJerk             = 0x0E,     // 減速度増加率設定

       CMD_ExpMode           = 0x60,     // 拡張モード設定
       CMD_HomeSpd           = 0x61,     // 原点検出速度設定
       CMD_SyncMode          = 0x64,     // 同期動作モード設定

       // データ読み出し命令
       CMD_ReadLp            = 0x10,     // 論理位置カウンタ読み出し
       CMD_ReadEp            = 0x11,     // 実位置カウンタ読み出し
       CMD_ReadSpeed         = 0x12,     // 現在ドライブ速度読み出し
       CMD_ReadAccDec              = 0x13,     // 現在加/減速度読み出し
       CMD_ReadSyncBuff            = 0x14,     // 同期バッファレジスタ読み出し

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

       // 補間命令
       CMD_IP_2ST            = 0x30,     // ２軸直線補間ドライブ
       CMD_IP_3ST            = 0x31,     // ３軸直線補間ドライブ
       CMD_IP_CW             = 0x32,     // ＣＷ円弧補間ドライブ
       CMD_IP_CCW            = 0x33,     // ＣＣＷ円弧補間ドライブ
       CMD_IP_2BP            = 0x34,     // ２軸ビットパターン補間ドライブ
       CMD_IP_3BP            = 0x35,     // ３軸ビットパターン補間ドライブ
       CMD_BP_ENABLED              = 0x36,     // ＢＰレジスタ書き込み可
       CMD_BP_DISABLED             = 0x37,     // ＢＰレジスタ書き込み不可
       CMD_BP_STACK          = 0x38,     // ＢＰデータスタック
       CMD_BP_CLR            = 0x39,     // ＢＰデータクリア
```

```
        CMD_IP_1STEP        = 0x3A,     // 補間シングルステップ
        CMD_IP_DEC_VALID            = 0x3B,     // 減速有効
        CMD_IP_DEC_INVALID  = 0x3C,     // 減速無効
        CMD_IP_INTRPT_CLR   = 0x3D,     // 補間割り込みクリア


        // その他の命令
        CMD_HOME_EXEC       = 0x62,     // 自動原点出し実行
        CMD_DEVCTR_CLR              = 0x63,     // 偏差カウンタクリアパルス出力
        CMD_SYNC_ACTIVE             = 0x65,     // 同期動作起動
        CMD_NOP                     = 0x0F,     // ＮＯＰ（軸切り換え用）
}
```

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

⑤補間終了メッセージ、終了ステータス　　※MCX314As搭載ボードのみ

[VC]

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

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

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

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

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

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

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

[C#]

```
    public enum MSG_ID : int
    {   // 補間終了メッセージ
        WM_USER                     =0x0400,
        WM_BP_END           =WM_USER+1,   // (WM_USER + 1) ＢＰ補間終了メッセージ
```

```
    WM_CIP_END          =WM_USER+2    // (WM_USER + 2) 連続補間終了メッセージ
}
```
（使用例）MSG_IDのWM_BP_ENDの場合は　　MSG_ID.WM_BP_END

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

    // ■補間開始前のエラー
    BP_CNT_ERR          = 0x111       ,      // 指定されたデータ数が範囲外
    BP_ALREADY_EXEC             = 0x112       ,      // 既にＢＰ補間、あるいは連続補間が実行中
    BP_THREAD_ERR       = 0x113       ,      // スレッドを起動できなかった
    BP_MALLOC_ERR       = 0x114       ,      // メモリを確保できなかった

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

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

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

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

```
public enum IP_AXIS : int
{   // 補間軸
    IP_X                =0,           // 補間軸Ｘ
    IP_Y                =1,           // 補間軸Ｙ
    IP_Z                =2,           // 補間軸Ｚ
    IP_U                =3            // 補間軸Ｕ
}
```
（使用例）補間軸Ｘを指定する場合は　　IP_AXIS.IP_X

⑥定数定義
［C#］

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

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

```
public enum MaxValue : uint
{
    MCX304              =268435455,   // MCX304の出力パルスの最大値
    MCX314As                    =4294967295   // MCX314Asの出力パルスの最大値
}
```
（使用例）MCX304の出力パルスの最大値を指定する場合は　　MaxValue.MCX304

⑦ＩＣ名

[C#]

```
public enum IC : int
{
    A                   =0,           // 1つ目のIC
    B                   =1,           // 2つ目のIC
    MCX304              =0,           // MCX304
    MCX314AS                    =1            // MCX314AS
}
```
（使用例）ＩＣ　Ａを指定する場合は　　IC.A

(2) 軸指定の方法は、下記の通りです。

軸	VC	C#
X	AXIS_X	AXIS.X
Y	AXIS_Y	AXIS.Y
Z	AXIS_Z	AXIS.Z
U	AXIS_U	AXIS.U
全軸	AXIS_ALL	AXIS.ALL

①１軸指定の場合

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

（使用例）　Ｘ軸にドライブ速度 1000 を設定する

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

②２軸指定の場合

ビットＯＲ演算子を使用します。

例えばＸ軸とＹ軸を一度に指定する場合、

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

（使用例）　Ｘ,Ｙ軸にドライブ速度 1000 を設定する

```
[VC]  Nmc_Speed(No, IcNo, AXIS_X | AXIS_Y, 1000);
[VB]  Call Nmc_Speed(No, IcNo, AXIS_X Or AXIS_Y, 1000)
[C#]  MC8043P.Nmc_Speed(No, IcNo, AXIS.X | AXIS.Y, 1000);
```

③３軸指定の場合

ビットＯＲ演算子を使用します。

例えばＸ軸とＹ軸とＺ軸を一度に指定する場合、

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

（使用例）　Ｘ,Ｙ,Ｚ軸にドライブ速度 1000 を設定する

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

④全軸指定の場合
AXIS_ALL を指定します。

（使用例） 全軸にドライブ速度 1000 を設定する
[VC] Nmc_Speed(No, IcNo, AXIS_ALL, 1000);
[VB] Call Nmc_Speed(No, IcNo, AXIS_ALL, 1000)
[C#] MC8000P.Nmc_Speed(No, IcNo, AXIS.ALL, 1000);

(3) 補間関数で使用する構造体（VBではユーザー定義型）の定義は、下記の通りです。 ※MCX314As搭載ボードのみ

①ＶＣ

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

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

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

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

②ＶＢ

```
’ ２軸ＢＰ補間
Type DATA_2BP
    Bp1p As Integer         ’BP1Pデータ
    Bp1m As Integer         ’BP1Mデータ
    Bp2p As Integer         ’BP2Pデータ
    Bp2m As Integer         ’BP2Mデータ
End Type

’ ３軸ＢＰ補間
Type DATA_3BP
    Bp1p As Integer         ’BP1Pデータ
    Bp1m As Integer         ’BP1Mデータ
    Bp2p As Integer         ’BP2Pデータ
    Bp2m As Integer         ’BP2Mデータ
    Bp3p As Integer         ’BP3Pデータ
    Bp3m As Integer         ’BP3Mデータ
End Type

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

’ ３軸連続補間
Type DATA_3CIP
    EndP1 As Long           ’終点（第１軸）
    EndP2 As Long           ’終点（第２軸）
    EndP3 As Long           ’終点（第３軸）
    Speed As Integer        ’速度（速度を変更する場合は１～８０００、変更しない場合は０をセットする）
End Type                    ’注：第１軸、第２軸、第３軸は、WR5で指定する
```

③ＶＢ．ＮＥＴ

```
’２軸ＢＰ補間
Structure DATA_2BP
     Dim Bp1p As Short       ’BP1Pデータ
     Dim Bp1m As Short       ’BP1Mデータ
     Dim Bp2p As Short       ’BP2Pデータ
     Dim Bp2m As Short       ’BP2Mデータ
End Structure

’３軸ＢＰ補間
Structure DATA_3BP
     Dim Bp1p As Short       ’BP1Pデータ
     Dim Bp1m As Short       ’BP1Mデータ
     Dim Bp2p As Short       ’BP2Pデータ
     Dim Bp2m As Short       ’BP2Mデータ
     Dim Bp3p As Short       ’BP3Pデータ
     Dim Bp3m As Short       ’BP3Mデータ
End Structure

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

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

④Ｃ＃．ＮＥＴ

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

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

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

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

構造体の使用例を以下に示します。

例１．定義、初期化が別の場合

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

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

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

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

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

例２．定義と初期化を同時に行う場合

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

(4) 補間関数で指定する補間軸(IpAxis)の指定は下記の通りです。　　※MCX314As搭載ボードのみ
４軸Ｘ、Ｙ、Ｚ、Ｕの中から補間する軸の組み合わせを、次の１６ビットデータの下位６ビットに設定します。２軸補間の場合は
第１軸、第２軸に、第３軸補間の場合は第１軸、第２軸、第３軸に設定します。

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

第３軸　　第２軸　　第

１軸

◆各ビットの説明

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

軸	コード(２進)
X	００
Y	０１
Z	１０
U	１１

第１軸：Ｘ、第２軸：Ｙ、第３軸：Ｚの例
D5　D4　D3　D2　D1　D0
1　0　0　1　0　0

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

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

◆Ｃ＃．ＮＥＴの場合、次の既定値を使い補間軸を指定して下さい。

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

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

補間軸の指定方法は、次の例のように、ビット演算を用いて設定します。

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

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

(5) 連続補間関数実行時の速度変更について　　※MCX314As搭載ボードのみ

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

◆各セグメント毎の速度の設定方法

DATA_2CIPのSpeed、あるいはDATA_3CIPのSpeed に、各セグメントの速度を設定します。
・前のセグメントと異なる速度にする場合は、1～8000 の速度を設定します。
・前のセグメントと同じ速度のままにする場合は、0 を設定します。

◆速度を変更するタイミングについて

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

１つ目のセグメントについては、セグメント実行前に速度を設定します。
２つ目以降のセグメントについては、該当セグメントが開始した直後(その次のセグメントが書き込み可能になった時)に
速度を設定します。
例えば、２つ目が開始し３つ目のセグメントが書き込み可能になった時に、２つ目のセグメントの速度を設定します。
この為、２つ目が開始しても２つ目の速度を設定するまでの間は１つ目の速度で実行されますので、ご注意下さい。

(6) アドレスの指定

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

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

ボード名	ライトレジスタのアドレス	リードレジスタのアドレス	MSM82C55のアドレス	PIX132のアドレス
MC8082P	０～Ｆ	０～Ｆ	-----	１４～１６
MC8042P	０～Ｆ	０～Ｆ	-----	１４～１６
MC8022P	０～Ｆ	０～Ｆ	-----	１４～１６
MC8080P	０～Ｆ	０～Ｆ	１０～１２	-----
MC8043P	０～７	０～７	-----	-----
MC8082Pe	０～Ｆ	０～Ｆ	-----	１４～１６
MC8043Pe	０～７	０～７	-----	-----

注１：アドレスは１６進数で記載しています。
注２：ライトレジスタ、リードレジスタにアクセスする場合、Nmc_OutPort,Nmc_InPort関数ではなく、専用の関数を使用した　　方が便利です。

(7) ＩＣ番号の指定は下記の通りです。

①ＩＣを１つ搭載しているボードの場合は０を指定します。

②ＩＣを２つ搭載しているボードの場合は次の値を指定します。
・ＩＣ－Ａの場合は０
・ＩＣ－Ｂの場合は１

各ボードの搭載ＩＣ数と指定するＩＣ番号は次の通りです。

ボード名	搭載ＩＣ数	ＩＣ番号
MC8082P	２	０，１
MC8042P	１	０
MC8022P	１	０
MC8080P	２	０，１
MC8043P	１	０
MC8082Pe	２	０，１
MC8043Pe	１	０

注：ＩＣとはMCX314As, MCX304などのＩＣの事を指します。

3.4.3　使用方法

■API関数宣言

API関数宣言は、下記の場所で行っています。

VC++、VC++.NET　　：MC8000P_DLL.H
VB6.0　　：MC8000P_DLL.bas
VB.NET2003、VB2005、VB2008　　：MC8000P_DLL.vb
C#.NET　　：取扱説明書または入力支援を利用してください。

■使用方法

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

■ボード番号について

アプリケーションから指定するボード番号は下記の通りです。

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

■関数使用時の注意

(1) VC,VB,C#（全ての言語）について

①Nmc_Open関数実行前に各関数を実行した場合の動作保証はできません。

②接続していないボードの番号を誤って指定した場合も､各関数の動作の保証はできません。

③１枚のボードに対して、２つ以上のアプリケーションから同時にアクセス（オープンなど）を行わないで下さい。

(2) VC,C#のみ

①割り込み処理関数を使用する場合はWindowsの性格上、割り込み発生からユーザー定義ルーチンへ制御が移行するまでの時間を保証することは出来ません。

②割込みを行う場合は、割込みユーザー関数（Nmc_SetEventで指定した関数）を実行中にクローズ処理（Nmc_Close又はNmc_CloseAll）を実行しないで下さい。
クローズ処理を行う場合は、必ず、割込みユーザー関数が終了している状態で行って下さい。

■ＶＣにて割込みを処理する方法

①Nmc_Open関数にて割込みを使用する設定にします。

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

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

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

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

■割込みユーザー関数例

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

④割込み処理するユーザー関数の設定を解除する場合は Nmc_ResetEvent を実行して下さい。
この関数を実行すると、ボードで割込みが発生してもユーザー関数は呼び出されません。

```
Nmc_ResetEvent(No);
```

■Ｃ＃にて割込みを処理する方法（Ｃ＃のみ）

①MC8000P.Nmc_SetEventにて、割込みを処理するメソッドを設定します。引数は指定できません。
複数枚ボードを使用する場合は、下記のように設定します。

（ボード番号０の場合）

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

（ボード番号１の場合）

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

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

（ボード番号０の割込みユーザー関数）

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

（ボード番号１の割込みユーザー関数）

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

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

■連続補間について　　※MCX314As搭載ボードのみ

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

注１：Nmc_2CIPExec, Nmc_3CIPExec, Nmc_2CIPExec_BG, Nmc_3CIPExec_BG

連続補間関数を使用する場合の注意事項：

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

■ＢＰ補間（ビットパターン補間）について　　※MCX314As搭載ボードのみ

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

注２：Nmc_2BPExec, Nmc_3BPExec, Nmc_2BPExec_BG, Nmc_3BPExec_BG

ＢＰ補間関数を使用する場合の注意事項：

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

■補間関数使用時の注意　　※MCX314As搭載ボードのみ

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

Nmc_2BPExec	Nmc_2BPExec_BG	Nmc_2CIPExec	Nmc_2CIPExec_BG
Nmc_3BPExec	Nmc_3BPExec_BG	Nmc_3CIPExec	Nmc_3CIPExec_BG

(2) 上記の補間関数実行中は、下記の処理を実行しないで下さい。

①補間命令（30h～3Dh）の実行
②ＷＲ５補間モードレジスタの変更

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

Nmc_2BPExec_BG	Nmc_2CIPExec_BG
Nmc_3BPExec_BG	Nmc_3CIPExec_BG

(4) 補間関数にて補間実行中、速度が速い場合は次のデータセットが間に合わず、補間が停止する場合があります。
比較的安定して補間停止が起こらない条件を測定しました。
補間関数実行中にアプリケーションを切り替えたり、何かイベントが発生する場合と発生しない場合では停止する速度が異なります。また、連続補間は、１回の移動量によって、停止する速度は変わってきます。

測定結果：
次の条件では、下記ドライブ速度の場合、補間が停止しませんでした。
それを超えるドライブ速度の時は、補間が停止した事がありました。

［測定環境］
ＯＳ　　：WindowsXP SP1
ＣＰＵ　：Celeron(R) CPU 2.53 GHz

◆　実行関数　：　Nmc_2BPExec

①補間関数実行中にアプリケーションを切り替えた場合

補間データ数　　：100
安定したドライブ速度　　：70 PPS

②補間関数実行中にウィンドウを全く触らなかった場合

補間データ数　　：1,000
安定したドライブ速度　　：600 PPS

◆　実行関数　：　Nmc_2CIPExec

①補間関数実行中にアプリケーションを切り替えた場合

１回の移動量　　：1,000パルス
補間データ数　　：100
安定したドライブ速度　　：5,000PPS

②補間関数実行中にウィンドウを全く触らなかった場合

１回の移動量　　：1,000パルス
補間データ数　　：500
安定したドライブ速度　　：30,000 PPS

■マルチスレッドアプリケーション開発時の注意

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

Nmc_xxx の関数では、軸切り替え処理、ＷＲ６，ＷＲ７にデータを書き込む処理、ＲＲ６，ＲＲ７にデータを読み出す処理を実行している関数があります。それぞれの Nmc_xxx 関数は次の通りです。

◆軸切り替え処理を実行している関数

Nmc_Reset Nmc_Command Nmc_Command_IP
Nmc_WriteReg0 Nmc_WriteReg1 Nmc_WriteReg2 Nmc_WriteReg3
Nmc_ReadReg1 Nmc_ReadReg2

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

Nmc_ReadLp Nmc_ReadEp Nmc_ReadSpeed Nmc_ReadAccDec Nmc_ReadSyncBuff

Nmc_2BPExec Nmc_3BPExec Nmc_2BPExec_BG Nmc_3BPExec_BG
Nmc_2CIPExec Nmc_3CIPExec Nmc_2CIPExec_BG Nmc_3CIPExec_BG

Nmc_WriteRegSetAxis Nmc_ReadRegSetAxis Nmc_WriteData Nmc_WriteData2 Nmc_ReadData

◆ＷＲ６，ＷＲ７にデータを書き込む処理を実行している関数

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

◆ＲＲ６，ＲＲ７にデータを読み出す処理を実行している関数

Nmc_ReadLp Nmc_ReadEp Nmc_ReadSpeed Nmc_ReadAccDec Nmc_ReadSyncBuff Nmc_ReadData

ＷＲ１～ＷＲ３書き込み、ＲＲ１～ＲＲ２読み出し、データ書き込み命令、データ読み出し命令を実行する場合、基本的には次の Nmc_xxx 関数を使用して下さい。

◆ＷＲ１～ＷＲ３書き込み

Nmc_WriteReg1 Nmc_WriteReg2 Nmc_WriteReg3 Nmc_WriteRegSetAxis

◆ＲＲ１～ＲＲ２読み出し

Nmc_ReadReg1 Nmc_ReadReg2 Nmc_ReadRegSetAxis

◆データ書き込み命令

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

◆データ読み出し命令

Nmc_ReadLp Nmc_ReadEp Nmc_ReadSpeed Nmc_ReadAccDec Nmc_ReadSyncBuff Nmc_ReadData

ＷＲ１～ＷＲ３書き込み、ＲＲ１～ＲＲ２読み出し、データ書き込み命令、データ読み出し命令を実行する際、これらの関数を使用せずに同じ処理を行った場合、マルチスレッド環境では注意が必要です。

（１）例えば、ＷＲ１書き込み時には Nmc_WriteReg1 を使用しますが、それ以外の方法としては下記の方法などがあります。

```
① Nmc_OutPort(No, MCX_WR0,  0x010F);          // Ｘ軸に切り替える（ＩＣ－Ａ）
② Nmc_OutPort(No, MCX_WR1,  Data);            // ＷＲ１書き込み（ＩＣ－Ａ）
```

また、次のような関数でも同じ処理を実行できます。

```
③ Nmc_WriteReg(No, IcNo, MCX_WR0,  0x010F); // Ｘ軸に切り替える
④ Nmc_WriteReg(No, IcNo, MCX_WR1,  Data);    // ＷＲ１書き込み
```

この場合、①と②の間、③と④の間に、軸を切り替える Nmc_xxx 関数が実行された場合、別の軸のＷＲ１にデータが書かれてしまいます。

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

```
① Nmc_OutPort( No, MCX_WR6, Data );     // ＷＲ６書き込み（ＩＣ－Ａ）
② Nmc_OutPort( No, MCX_WR0, 0x0105 );   // ＷＲ６データをＸ軸の速度に設定する（ＩＣ－Ａ）
```

また、次のような関数でも同じ処理を実行できます。

```
③ Nmc_WriteReg6(No, IcNo, Data);              // ＷＲ６書き込み
④ Nmc_Command(No, IcNo, AXIS_X, 0x05);// ＷＲ６データをＸ軸の速度に設定する
```

この場合、①と②の間、③と④の間に、ＷＲ６，ＷＲ７にデータを書き込む Nmc_xxx 関数が実行された場合、別のデータが速度に書かれてしまいます。

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

```
① Nmc_OutPort( No, MCX_WR0, 0x0110 );   // Ｘ軸の論理位置カウンタをＲＲ６，ＲＲ７に読み出す（ＩＣ－Ａ）
② d6 = Nmc_InPort( No, MCX_RR6 );       // ＲＲ６を読み出す（ＩＣ－Ａ）
③ d7 = Nmc_InPort( No, MCX_RR7 );       // ＲＲ７を読み出す（ＩＣ－Ａ）
```

この場合、①と②の間、②と③の間に、ＲＲ６，ＲＲ７にデータを読み出す Nmc_xxx 関数が実行された場合、ここでは別のデータが読み出されてしまいます。

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

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

3.5　プログラミング上の注意点

（１）入力信号フィルタの初期設定

■MCX314As搭載ボードの場合

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

フィルタ遅延時間：512μsec

各信号のフィルタの有効/無効：

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

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

(例１)
```
  Nmc_ExpMode(0, 0, AXIS_ALL, 0x5F00, 0x0000);  // ＩＣ－Ａの設定
  Nmc_ExpMode(0, 1, AXIS_ALL, 0x5F00, 0x0000);  // ＩＣ－Ｂの設定（複数ＩＣの場合）
```

(例２)
```
  // ＩＣ－Ａの設定
  Nmc_WriteReg6(0, 0, 0x5F00);
  Nmc_WriteReg7(0, 0, 0x0000);
  Nmc_WriteReg0(0, 0, 0x0F60);

  // ＩＣ－Ｂの設定（複数ＩＣの場合）
  Nmc_WriteReg6(0, 1, 0x5F00);
  Nmc_WriteReg7(0, 1, 0x0000);
  Nmc_WriteReg0(0, 1, 0x0F60);
```

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

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

■MCX304搭載ボードの場合

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

フィルタ遅延時間：512μsec

各信号のフィルタの有効/無効：

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

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

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

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

（２）パソコンのスタンバイ、休止動作

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

（３）割り込みサポートについて

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

本デバイスドライバでサポートしている割り込みの種類は下記の通りです。

■MCX304搭載ボード
・ＲＲ３レジスタで報告される割り込み全て

■MCX314As搭載ボード
・ＲＲ３レジスタで報告される割り込み全て
・連続補間ドライブで次セグメントのデータと補間ドライブ命令が書き込み可能となった時の割り込み
・ビットパターン補間において、スタックカウンタの値が２から１に変わった時の割り込み

（４）割り込みクリアについて

①ＲＲ３レジスタで報告される割り込みについて

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

②連続補間ドライブで次セグメントのデータと補間ドライブ命令が書き込み可能となった時の割り込みについて
(MCX314As搭載ボードのみ)

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

③ビットパターン補間において、スタックカウンタの値が２から１に変わった時の割り込みについて
(MCX314As搭載ボードのみ)

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

（５）RR3割り込みと補間割り込みを両方使用する場合（MCX314As搭載ボードのみ）

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

例）割り込み発生時の割り込みユーザー関数処理
①Nmc_ReadEventにてRR3の割り込み要因を読み出し、RR3で割り込みが発生しているか確認する。
②補間割り込みが発生しているか確認する。(RR0のCNEXT、またはRR0のBPSC1,0を確認)

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

注２：Nmc_SetEvent関数で指定したユーザー関数

４．評価ツール

評価ツールプログラムは、MCX304 評価ツールとMCX314As 評価ツールの２種類です。

(1) MCX304 評価ツール・・・・MCX304搭載ボード(MC8082P, MC8080P, MC8082Peなど)を評価するツールです。(4.1参照)
(2) MCX314As 評価ツール・・・MCX314As搭載ボード(MC8043P, MC8043Peなど)を評価するツールです。(4.2参照)

4.1　MCX304評価ツール

MCX304 評価ツールプログラムは、MCX304を搭載しているボード(MC8082P, MC8080P, MC8082Peなど)を評価するツールです。
弊社ホームページよりダウンロードすることができます。（MC8000Pデバイスドライバソフトに添付）
評価ツールを実行する前に、MC8000Pデバイスドライバをインストールして下さい。

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

4.1.1実行プログラムについて

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

■評価するMCX304について

各評価ツールが評価するMCX304は下記の通りです。

●MCX304-A.exe

①MCX304を１つ搭載しているボード(MC8042P, MC8022Pなど)のMCX304
②MCX304を２つ以上搭載しているボード(MC8082P, MC8080P, MC8082Peなど)の1つ目のMCX304 （ MCX304-A ）

●MCX304-B.exe

①MCX304を２つ以上搭載しているボード(MC8082P, MC8080P, MC8082Peなど)の２つ目のMCX304 （ MCX304-B ）

■実行について

同じボードに対してMCX304-A.exeとMCX304-B.exeを同時に実行する事はできます。
但し、その場合は割込みを使用(設定)しないで下さい。
１つのボードに対してMCX304-A.exeのみ、またはMCX304-B.exeのみを実行する場合は、割込みを使用する事ができます。

異なるボードに対してMCX304-A.exe、またはMCX304-B.exeを同時に実行する事はできます。
その場合は、割込みを使用する事ができます。

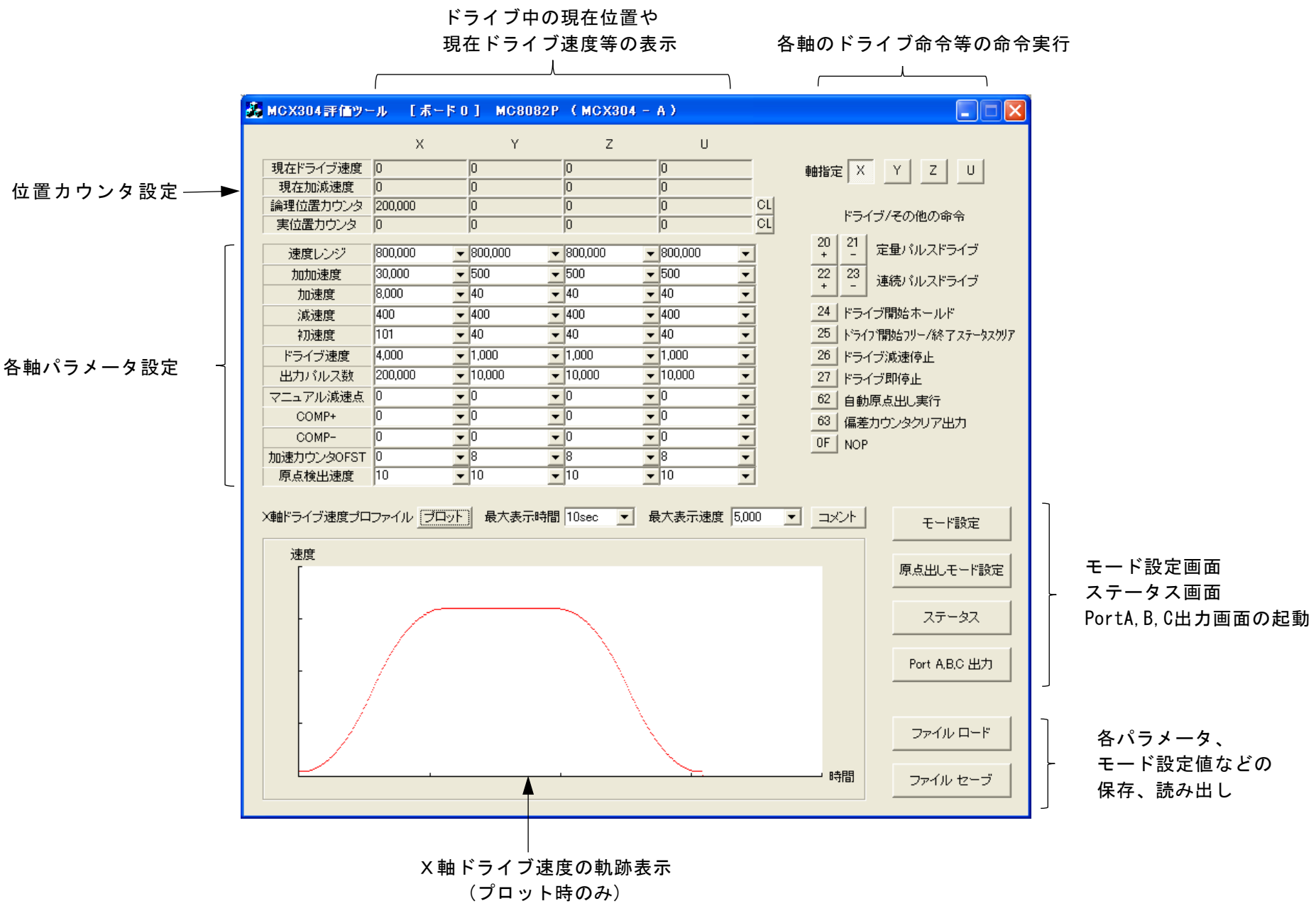
4.1.2機能概要

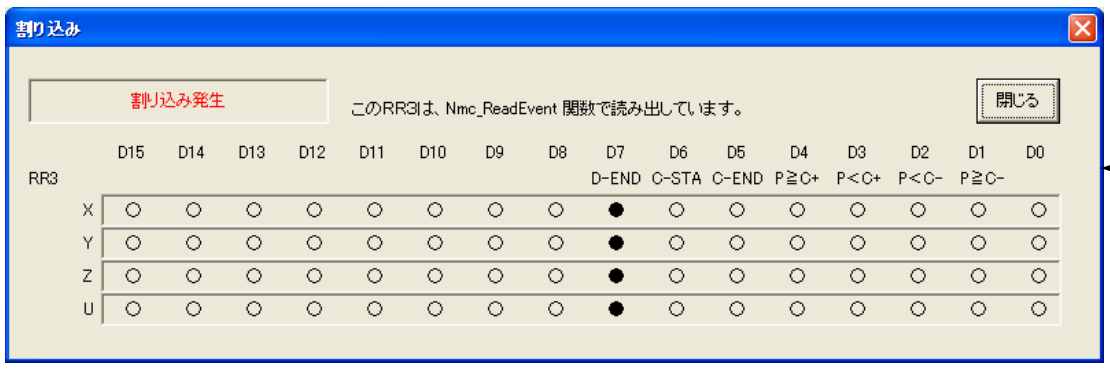
評価ツールを起動すると、ボード番号選択画面が表示され、ボード番号（ボードのロータリースイッチ番号(0～F)）を選択することができます。ボード名表示ボタンを押すと、ボード番号の横にボード名(本ドライバを使用しているボードのみ)を表示することができます。ボード番号を選択後、ＯＫボタンを押すと、メイン画面が表示されます。

メイン画面では、各軸パラメータ設定、ドライブ命令等の命令実行、現在位置・現在速度の表示、割り込み画面表示、パラメータ・モード設定値の保存、読み出し等を行います。また、モード設定画面やステータス画面を開き、モード設定、ステータス参照等を行う事ができます。MC8082P、またはMC8080Pの場合は、Port A,B,C 出力画面でポートA,B,Cの汎用出力を行う事ができます。

4.1.3 メイン画面

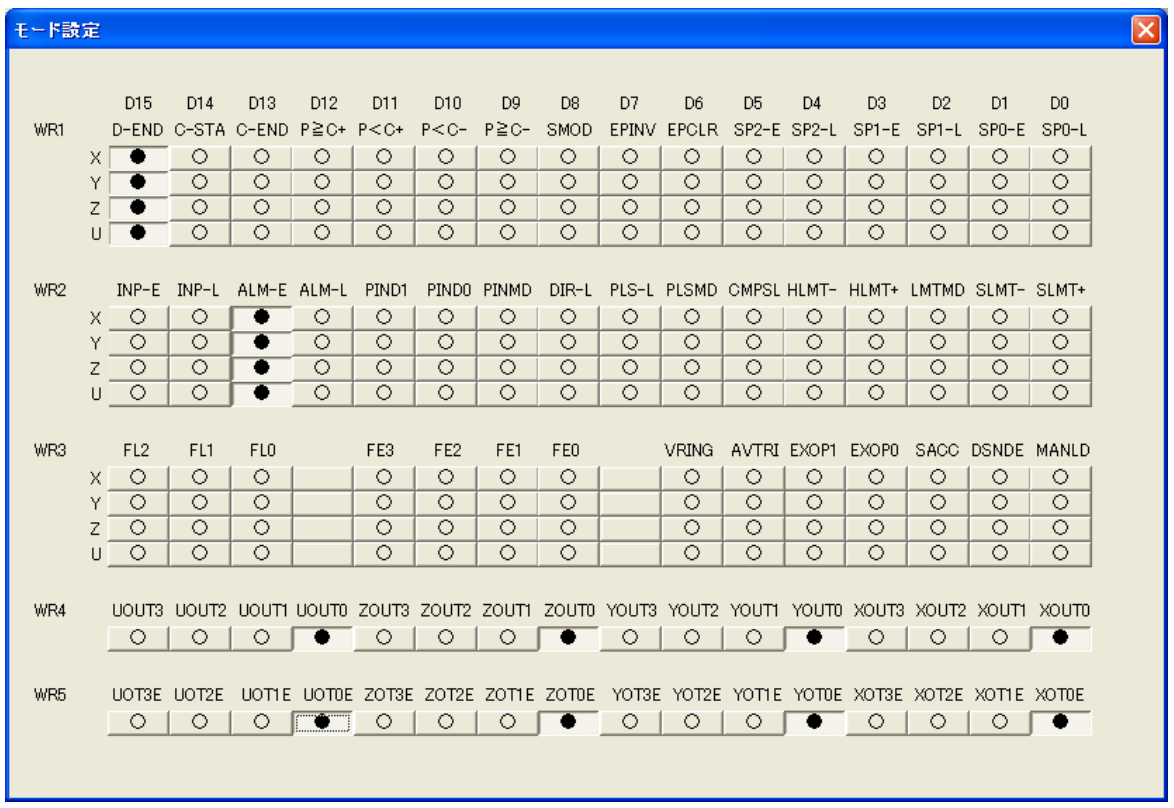
メイン画面では、下図のような操作を行います。





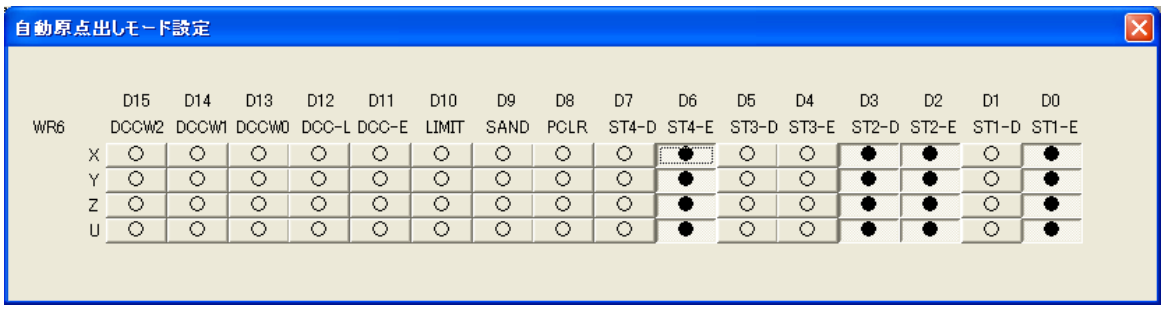
4.1.4 モード設定画面

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



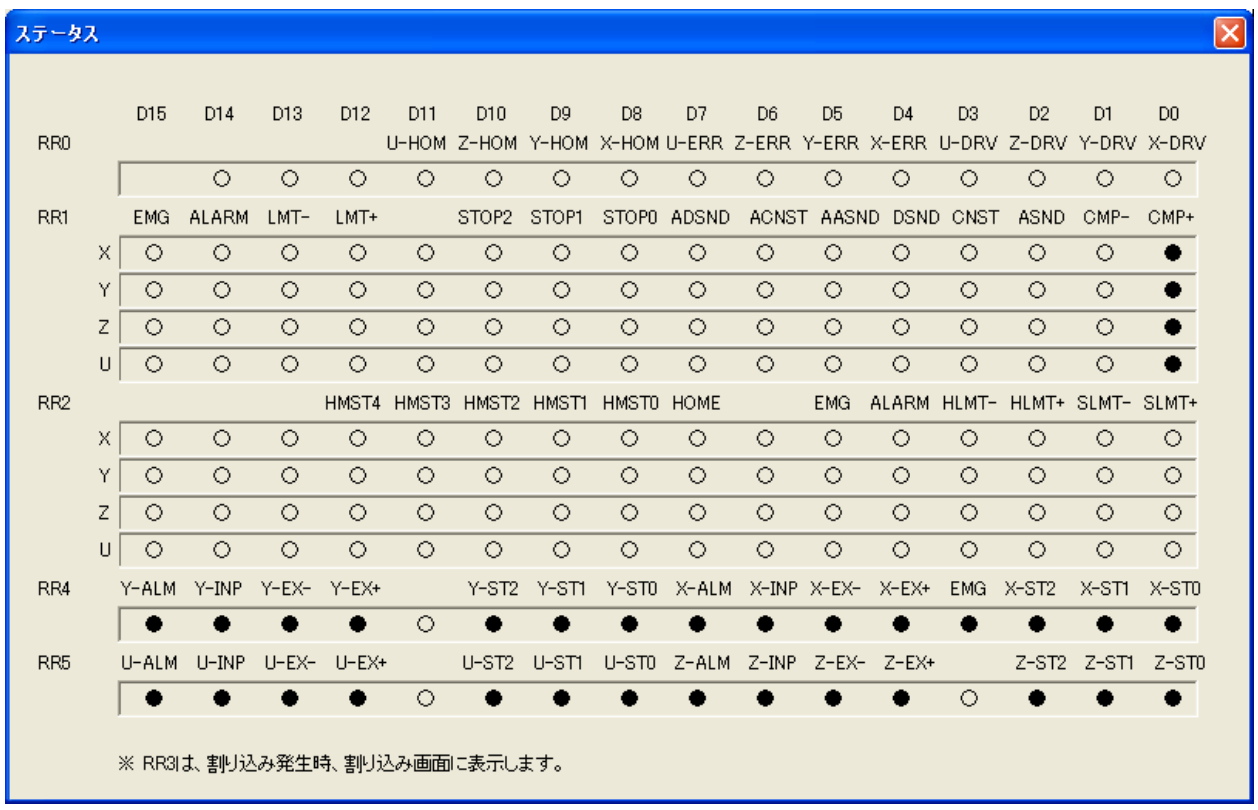
4.1.5 自動原点出しモード設定画面

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



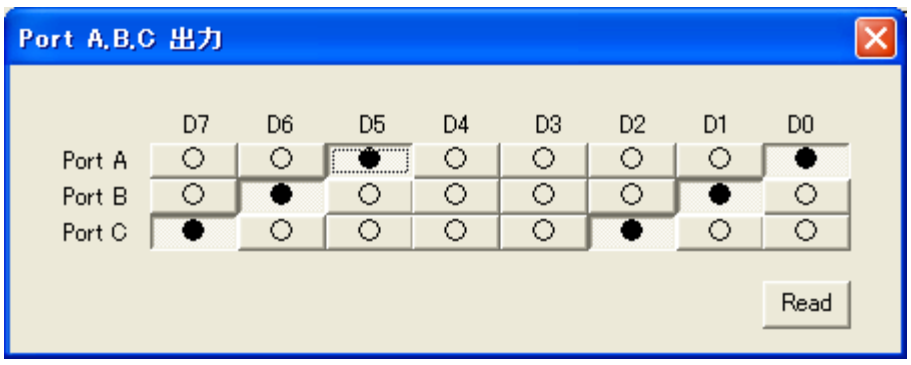
4.1.6 ステータス画面

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



4.1.7 Port A,B,C 出力画面

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



4.2　MCX314As評価ツール

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

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

4.2.1 実行プログラムについて

実行プログラムはMCX314As-A.exeとMCX314As-B.exeの２種類です。

■評価するMCX314Asについて

各評価ツールが評価するMCX314Asは下記の通りです。

●MCX314As-A.exe

①MCX314Asを１つ搭載しているボード（MC8043P, MC8043Peなど）のMCX314As
②MCX314Asを２つ以上搭載しているボードの1つ目のMCX314As（ MCX314As-A ）

●MCX314As-B.exe

①MCX314Asを２つ以上搭載しているボードの２つ目のMCX314As（ MCX314As-B ）

■実行について

同じボードに対してMCX314As-A.exeとMCX314As-B.exeを同時に実行する事はできます。
但し、その場合は割込みを使用(設定)しないで下さい。
１つのボードに対してMCX314As-A.exeのみ、またはMCX314As-B.exeのみを実行する場合は、割込みを使用する事ができます。
異なるボードに対してMCX314As-A.exe、またはMCX314As-B.exeを同時に実行する事はできます。
その場合は、割込みを使用する事ができます。

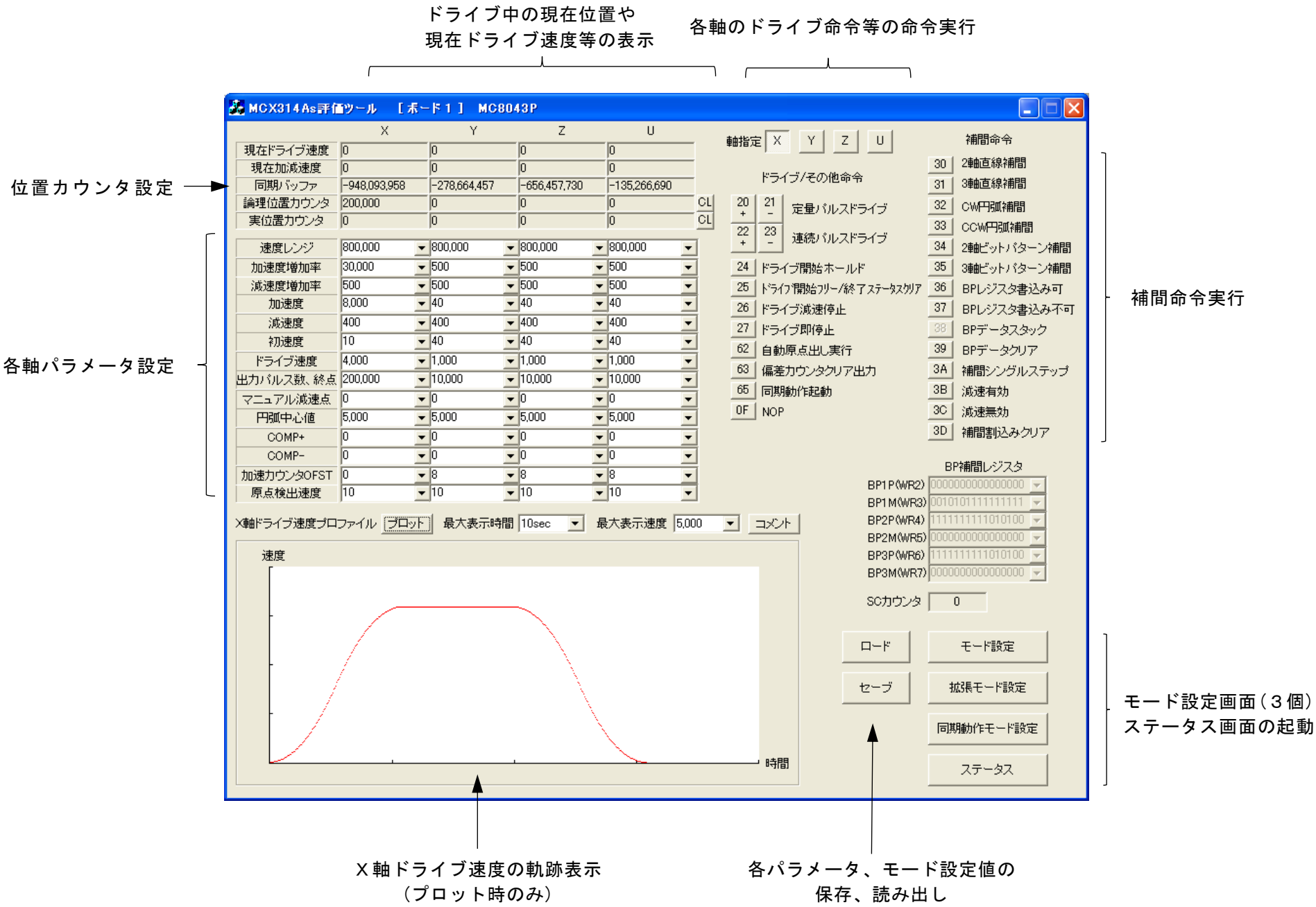
4.2.2 機能概要

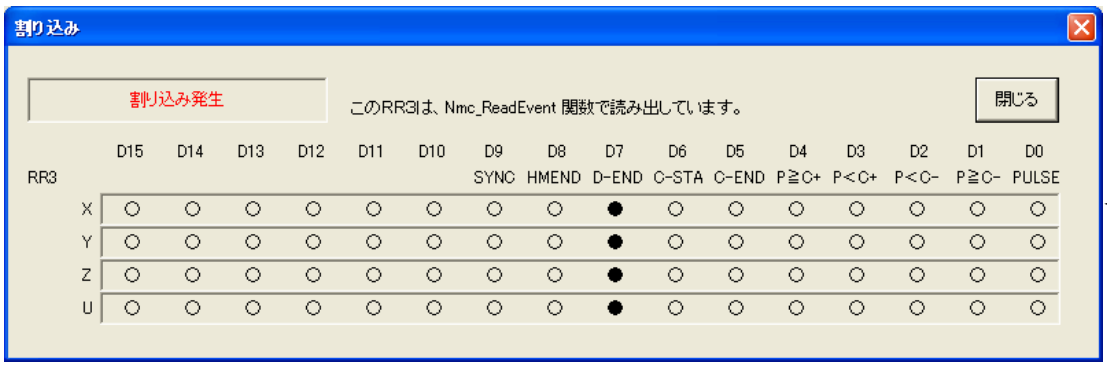
評価ツールを起動すると、ボード番号選択画面が表示され、ボード番号（ボードのロータリースイッチ番号(0～F)）を選択することができます。ボード名表示ボタンを押すと、ボード番号の横にボード名(本ドライバを使用しているボードのみ)を表示することができます。ボード番号を選択後、ＯＫボタンを押すと、メイン画面が表示されます。

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

4.2.3 メイン画面

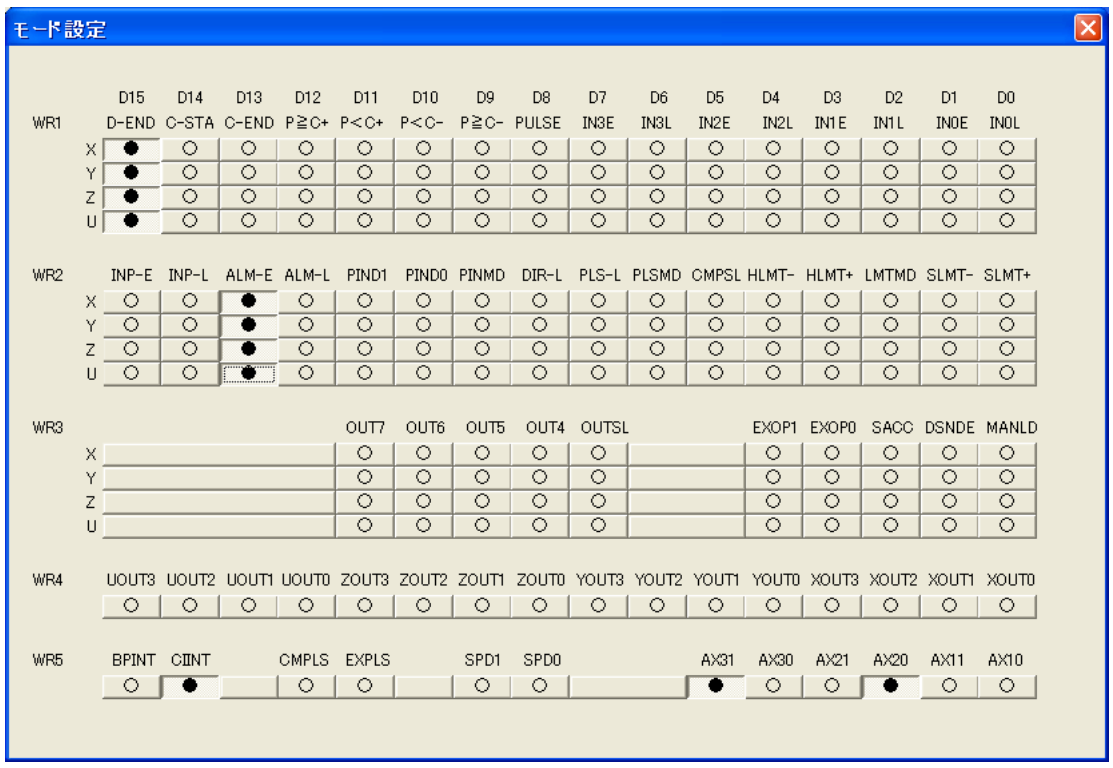
メイン画面では、下図のような操作を行います。





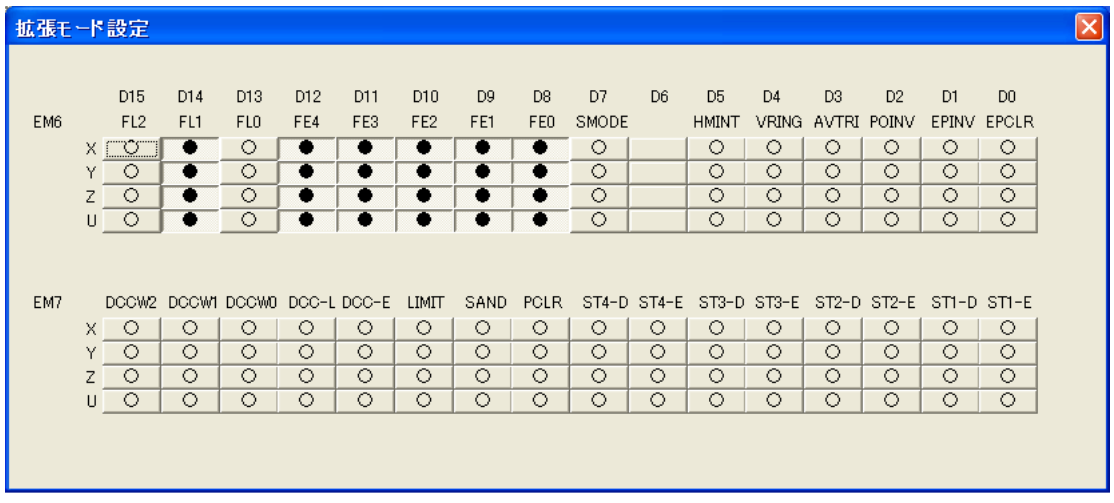
4.2.4 モード設定画面

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



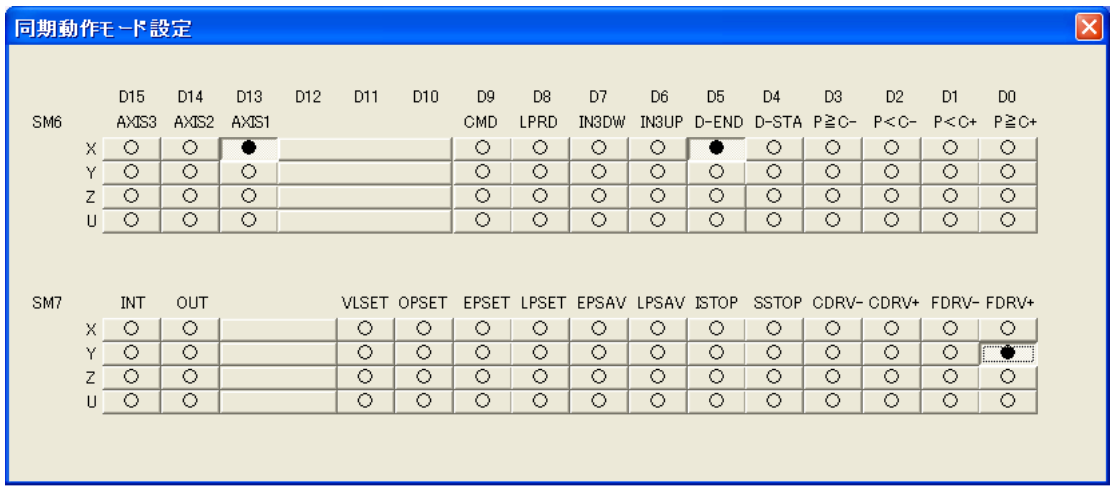
4.2.5 拡張モード設定画面

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



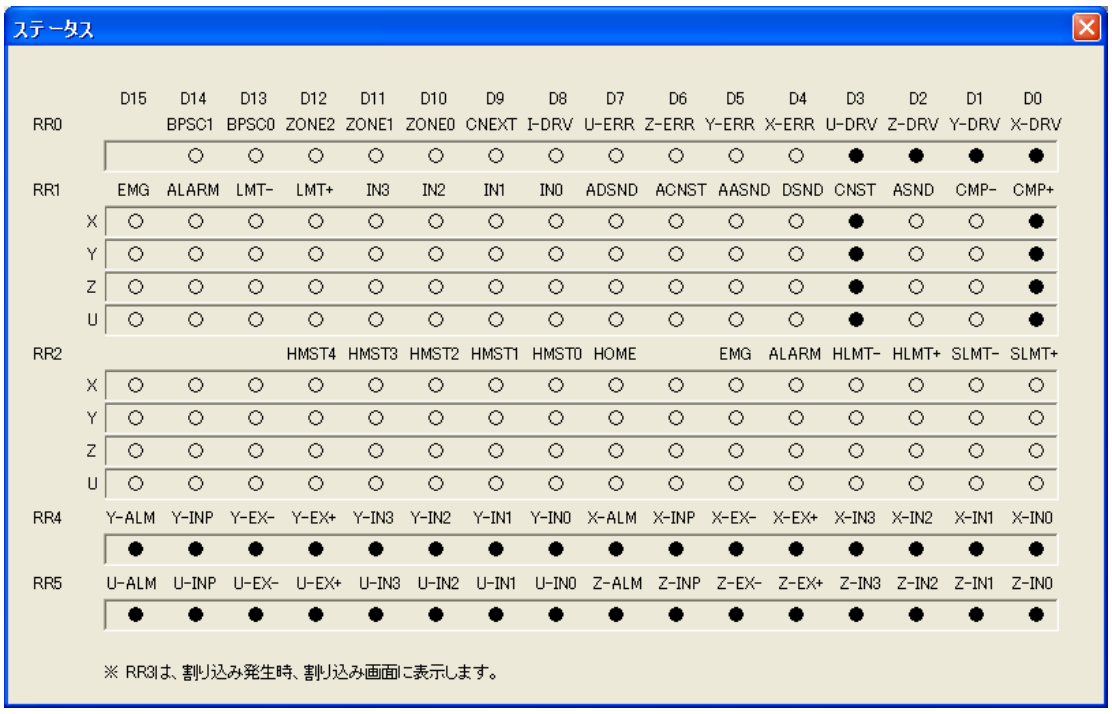
4.2.6 同期動作モード設定画面

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



4.2.7 ステータス画面

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



５．既存MC8080Pアプリケーションの移植

本章では、MC8080Pデバイスドライバ(MC8080P専用ドライバ)のＡＰＩ(OpenMC8080P関数など)を使用して既にMC8080P専用のアプリケーションを作成されたお客様が、本デバイスドライバ用にアプリケーションを移植する方法について説明します。

アプリケーションを移植することにより、既存のMC8080Pアプリケーション(OpenMC8080P関数などを使用したアプリケーション)から
MC8082Pボードを使用できるようになります。この場合、デバイスドライバは、本デバイスドライバを使用します。

移植方法は、次の通りです。

■移植方法

①「2. インストール」の説明に従い、MC8082Pボードをパソコンに挿入し、本デバイスドライバをインストールします。
②「5.1 アプリケーション変更方法」の説明に従い、既存のMC8080Pアプリケーションを変更します。

注意：本デバイスドライバの対応ＯＳは、Windows98 Windows2000 WindowsXPです。
WindowsMe, WindowsNT はサポートしていませんので、ご注意ください。

5.1　アプリケーション変更方法

5.1.1　ＶＣ＋＋アプリケーションの場合(VC++6.0, VC++.NET2003)

MC8080Pアプリケーションで使用しているMC8080P.libを今回提供のMC8000P.libに差し替え、リビルドする事で、MC8082Pボードにアクセスできるアプリケーションに変更する事ができます。変更方法は次の通りです。

① ¥Lib¥VC6フォルダに入っているMC8000P.lib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てください。

② VC++6.0 の場合は、[プロジェクト]－[設定]で「リンク」タブを選択し「オブジェクト／ライブラリモジュール」から
MC8080P.libを削除し、MC8000P.libを追加して下さい。（「図3.3-1 VC++6.0 プロジェクトの設定」を参照。）
VC++.NET2003 の場合は、[プロジェクト]－[プロパティ]画面で[リンカ]－[入力]を選択し、「追加の依存ファイル」から
MC8080P.libを削除し、MC8000P.libを追加して下さい。（「図3.3-2 VC++.NET2003 プロジェクトのプロパティ」を参照。）

③ 既存のMC8080P.libファイルは使用しませんので、別フォルダに移動しても構いません。

④ 5.2.1 VC++の関数をそのまま使用する事ができます。

⑤ アプリケーションをリビルドして下さい。
リビルドが正常終了した場合、MC8000Pデバイスドライバが正常にインストールされたマシンでアプリケーションを実行できます。

5.1.2　ＶＢ６.０アプリケーションの場合

MC8080Pアプリケーションで使用しているMC8080P_DLL.basを今回提供のMC8080P_DLL.basに差し替え、exeファイルを作成する事で、
MC8082Pボードにアクセスできるアプリケーションに変更する事ができます。変更方法は次の通りです。

① 既存のMC8080Pアプリケーションのフォルダにあ るMC8080P_DLL.bas（MC8080P用に提供された既存ファイル）を別フォルダに移動
してください。本デバイスドライバではこのファイルを使用しません。
但し、このMC8080P_DLL.basファイルに、既に追加修正を行っている場合は、追加した部分のみ残す必要があります。

② ¥Lib¥MC8080P_old¥VB6 フォルダに入っているMC8080P_DLL.basファイルをアプリケーションのフォルダにコピーしてください。

③ アプリケーションのプロジェクトからMC8080P_DLL.basファイルを開き、今回提供のMC8080P_DLL.basファイルであることを確認
してください。

④ 5.2.2 VB6.0の関数をそのまま使用する事ができます。

⑤ exeファイルを作成して下さい。
コンパイルが正常終了した場合、MC8000Pデバイスドライバが正常にインストールされたマシンでアプリケーションを実行できます。

5.1.3　ＶＢ.ＮＥＴ２００３アプリケーションの場合

MC8080Pアプリケーションで使用しているMC8080P_DLL.vbを今回提供のMC8080P_DLL.vbに差し替え、リビルドする事で、MC8082Pボードにアクセスできるアプリケーションに変更する事ができます。変更方法は次の通りです。

①　既存のMC8080Pアプリケーションのフォルダにあ るMC8080P_DLL.vb（MC8080P用に提供された既存ファイル）を別フォルダに移動
してください。本デバイスドライバではこのファイルを使用しません。
但し、このMC8080P_DLL.vbファイルに、既に追加修正を行っている場合は、追加した部分のみ残す必要があります。

②　¥Lib¥MC8080P_old¥VB.NET2003 フォルダに入っているMC8080P_DLL.vbファイルをアプリケーションのフォルダにコピーしてくだ
さい。

③　アプリケーションのプロジェクトからMC8080P_DLL.vbファイルを開き、今回提供のMC8080P_DLL.vbファイルであることを確認して　ください。

④　5.2.3 VB.NET2003の関数をそのまま使用する事ができます。

⑤　アプリケーションをリビルドして下さい。
リビルドが正常終了した場合、MC8000Pデバイスドライバが正常にインストールされたマシンでアプリケーションを実行できます。

5.2　関数仕様

MC8080Pのアプリケーションを既に開発されたお客様の為に、本デバイスドライバでは、下記のMC8080P関数をサポートしています。
下記関数を使用して、MC8082Pに対してアクセスする事が可能です。
また、MC8080Pボードのデバイスドライバを本デバイスドライバに変更した場合、下記関数を使用して、本デバイスドライバからMC8080Pに対してアクセスする事も可能です。

MC8082Pのアプリケーションを新規に開発される方は、3.4 APIの関数を使用して下さい。

5.2.1　ＶＣ＋＋

関数名	機能　及び　内容
OpenMC8080P	MC8080Pの使用を開始する。 入力パラメータ：int No // カード番号（カード上のロータリースイッチの設定値(0～15)） 戻り値　　：BOOL　// オープンに成功するとTRUE、失敗するとFALSE 使用例： 　status = OpenMC8080P(0);　　// カード番号０をオープン
CloseMC8080P	MC8080Pの使用を終了する。 入力パラメータ：int No // カード番号（カード上のロータリースイッチの設定値(0～15)） 戻り値　　：BOOL　// クローズに成功するとTRUE、失敗するとFALSE 使用例： 　status = CloseMC8080P(0);　　// カード番号０をクローズ
CloseAllMC8080P	全てのMC8080Pの使用を終了する。 入力パラメータ：なし 戻り値　　：なし 使用例： 　CloseAllMC8080P();　// 全てのカードをクローズ
OutpMC8080P	出力ポートに２バイトデータを書き込む。 入力パラメータ：int No　// カード番号（カード上のロータリースイッチの設定値(0～15)） 　　long Adr // 書き込むアドレス 　　long Dat // 書き込むデータ 戻り値　　：なし 使用例： 　OutpMC8080P(0, MCX304A_WR0, 0x8000);　// ソフトリセット
InpMC8080P	入力ポートから２バイトデータを読み出す。 入力パラメータ：int No　// カード番号（カード上のロータリースイッチの設定値(0～15)） 　　long Adr // 読み出すアドレス 戻り値　　：long　　// 入力ポートから読み込んだデータ 使用例： 　data = InpMC8080P(0, MCX304A_RR0);　// リードレジスタ RR0 の読み出し

SetEventMC8080P	割込みを処理するユーザー関数を設定する。
	
	入力パラメータ：int No　// カード番号（カード上のロータリースイッチの設定値(0～15)）
	　　Long Adr　//　関数のアドレス
	戻り値　　：なし
	使用例：
	　OutpMC8080P(CARD_NO, MCX304A_WR0, 0x0F0F); // 全軸指定
	　OutpMC8080P(CARD_NO, MCX304A_WR1, 0x8000); // 停止時割込み発生
	　SetEventMC8080P(CARD_NO,(LPTHREAD_START_ROUTINE)MC8080P_EventProc0);

関数名	機能　及び　内容
ResetEventMC8080P	割込みを処理するユーザー関数の設定を解除する。 入力パラメータ：int No　// カード番号（カード上のロータリースイッチの設定値(0～15)） 戻り値　　：なし 使用例： OutpMC8080P(CARD_NO, MCX304A_WR0, 0x0F0F);　// 全軸指定 OutpMC8080P(CARD_NO, MCX304A_WR1, 0x0000);　// 割込み禁止 ResetEventMC8080P(CARD_NO);
ReadEventMC8080P	割込み発生直後の各軸ＲＲ３の値を取得する。（ドライバ内のＲＲ３データは読み出し後クリアされる） 入力パラメータ：int No　　// カード番号（カード上のロータリースイッチの設定値(0～15)） long* Rr3AX　// ＡＸ軸のＲＲ３を格納する為のバッファへのポインタ long* Rr3AY　// ＡＹ軸のＲＲ３を格納する為のバッファへのポインタ long* Rr3AZ　// ＡＺ軸のＲＲ３を格納する為のバッファへのポインタ long* Rr3AU　// ＡＵ軸のＲＲ３を格納する為のバッファへのポインタ long* Rr3BX　// ＢＸ軸のＲＲ３を格納する為のバッファへのポインタ long* Rr3BY　// ＢＹ軸のＲＲ３を格納する為のバッファへのポインタ long* Rr3BZ　// ＢＺ軸のＲＲ３を格納する為のバッファへのポインタ long* Rr3BU　// ＢＵ軸のＲＲ３を格納する為のバッファへのポインタ 戻り値　　：なし 使用例： long Rr3AX,Rr3AY,Rr3AZ,Rr3AU,Rr3BX,Rr3BY,Rr3BZ,Rr3BU; ReadEventMC8080P(CARD_NO, &Rr3AX,&Rr3AY,&Rr3AZ,&Rr3AU,&Rr3BX,&Rr3BY,&Rr3BZ,&Rr3BU); 注意 ボードで割込が発生した直後ドライバ内でＲＲ３を読み出してしまうのでＲＲ３はクリアされてしまいます。割込み発生直後のＲＲ３を確認する場合はこの関数を使用してください。 また、SetEventMC8080P、ResetEventMC8080P関数の実行とは関係なく、割込みが発生するとドライバ　は必ずRR3データを読み出し保存します。ドライバ内に保存されたRR3データは、ReadEventMC8080P 関数を実行して読み出すとクリアされます。 ドライバ内のRR3データをクリアしたい時は、ReadEventMC8080P関数を実行して下さい。

5.2.2 ＶＢ６．０

関数名	機能　及び　内容
OpenMC8080P	MC8080Pの使用を開始する。 入力パラメータ：No As Long　'カード番号(カード上のロータリースイッチの設定値(0～15)) 戻り値　　：Long　　'オープンに成功すると0以外、失敗すると0 使用例： 　status = OpenMC8080P(0)　　'カード番号0をオープン
CloseMC8080P	MC8080Pの使用を終了する。 入力パラメータ：No As Long　'カード番号(カード上のロータリースイッチの設定値(0～15)) 戻り値　　：Long　　'クローズに成功すると0以外、失敗すると0 使用例： 　status = CloseMC8080P(0)　　'カード番号0をクローズ
CloseAllMC8080P	全てのMC8080Pの使用を終了する。 入力パラメータ：なし 戻り値　　：なし 使用例： 　CloseAllMC8080P　　'全てのカードをクローズ
OutpMC8080P	出力ポートに２バイトデータを書き込む。 入力パラメータ：No As Long　'カード番号(カード上のロータリースイッチの設定値(0～15)) 　　Adr As Long '書き込むアドレス 　　Dat As Long '書き込むデータ 戻り値　　：なし 使用例： 　Call OutpMC8080P(0, MCX304A_WR0, &H8000) 'ソフトリセット
InpMC8080P	入力ポートから２バイトデータを読み出す。 入力パラメータ：No As Long　'カード番号(カード上のロータリースイッチの設定値(0～15)) 　　Adr As Long '読み出すアドレス 戻り値　　：Long　　'入力ポートから読み込んだデータ 使用例： 　data = InpMC8080P(0, MCX304A_RR0)　'リードレジスタ RR0 の読み出し

5.2.3 ＶＢ.ＮＥＴ２００３

関数名	機能　及び　内容
OpenMC8080P	MC8080Pの使用を開始する。 入力パラメータ：No As Integer　'カード番号(カード上のロータリースイッチの設定値(0～15)) 戻り値　　：Integer　　'オープンに成功すると０以外、失敗すると０ 使用例： 　status = OpenMC8080P(0)　　' カード番号０をオープン
CloseMC8080P	MC8080Pの使用を終了する。 入力パラメータ：No As Integer　'カード番号(カード上のロータリースイッチの設定値(0～15)) 戻り値　　：Integer　　'クローズに成功すると０以外、失敗すると０ 使用例： 　status = CloseMC8080P(0)　　'カード番号０をクローズ
CloseAllMC8080P	全てのMC8080Pの使用を終了する。 入力パラメータ：なし 戻り値　　：なし 使用例： 　CloseAllMC8080P　　'全てのカードをクローズ
OutpMC8080P	出力ポートに２バイトデータを書き込む。 入力パラメータ：No As Integer　'カード番号(カード上のロータリースイッチの設定値(0～15)) 　　Adr As Integer '書き込むアドレス 　　Dat As Integer '書き込むデータ 戻り値　　：なし 使用例： 　Call OutpMC8080P(0, MCX304A_WR0, &H8000) 'ソフトリセット
InpMC8080P	入力ポートから２バイトデータを読み出す。 入力パラメータ：No As Integer　'カード番号(カード上のロータリースイッチの設定値(0～15)) 　　Adr As Integer '読み出すアドレス 戻り値　　：Integer　　'入力ポートから読み込んだデータ 使用例： 　data = InpMC8080P(0, MCX304A_RR0)　'リードレジスタ RR0 の読み出し

5.2.4　注意点

（１）開始・終了処理

各関数を使用する前にOpenMC8080P関数を必ず実行してください。
プログラム終了時にCloseMC8080P,またはCloseAllMC8080P関数を実行してください。

（２）割り込みについて

VC++のみで割り込みをサポートしていますが、割り込み処理関数を使用する場合はWindowsの性格上、割り込み発生から
ユーザー関数へ制御が移行するまでの時間を保証することはできません。

VC++で割込みを行う場合は、割込みユーザー関数（SetEventMC8080P関数で指定したユーザー関数）を実行中にクローズ処理
（CloseMC8080P,またはCloseAllMC8080P関数）を実行しないで下さい。
クローズ処理を行う場合は、必ず、割込みユーザー関数が終了している状態で行って下さい。

（３）ポートA,B,Cの汎用出力

ポートA,B,Cの汎用出力については、下表のようにMC8080PとMC8082PでI/Oアドレスが異なりますが、OutpMC8080P関数を使用
してこの汎用出力を行っている場合は、MC8000Pデバイスドライバが自動的にアドレスを変換していますので、アプリケーショ
ンを変更する必要はありません。

	MC8080P	MC8082P
ポートＡのアドレス	10	14
ポートＢのアドレス	11	15
ポートＣのアドレス	12	16

※アドレスは１６進数です。

MC8000Pデバイスドライバは、MC8082Pボードの10h～12hに対してOutpMC8080P関数によるデータ書き込み要求があった場合、
MC8082Pの14h～16hに書き込みます。（アドレス変換を行います。）

MC8080Pボードの10h～12hに対してOutpMC8080P関数によるデータ書き込み要求があった場合は、そのままMC8080Pの10h～12hに
書き込みます。（アドレス変換は行いません。）

（４）入力信号フィルタの初期設定

「3.5　プログラミング上の注意点」の「（１）入力信号フィルタの初期設定　■MCX304搭載ボードの場合」を参照して
ください。

（５）その他

「3.5　プログラミング上の注意点」の章を参照してください。