ADLINK TECHNOLOGY INC.

PCI-8158
高密度高機能
8 軸サーボ / ステッパ
モーション・コントロール・カード
ユーザー・マニュアル

マニュアルバージョン2.00

改訂日 : 2007 年 12 月 5 日

品番 : 50-11139-1000

Recycled Paper

技術革新;世界のオートメーション化

本書の情報は信頼性、デザイン、機能などの改善で予告なしに変更されることがあります。また、製造元の側のコミットメントを示すものではありません。

製造元は、本製品または本書の使用または不使用によって発生したいかなる直接的、間接的、特別の、付随的、または結果的な損害に対して、たとえこのような損害が生じる可能性について報告を受けていたとしても、一切責任を負いません。

商標

NuDAQ、NuIPC、DAQBench は ADLINK TECHNOLOGY 社の登録商標です。

本書に記載されている製品名は認証目的のためだけで、各社の商標または登録商標となっている場合があります。

ADLINK にサービスを申し込む

お客様の満足は ADLINK Technology Inc. の最優先事項です。サービスやサポートを必要とする場合はお知らせください。

ADLINK TECHNOLOGY INC.

ウェブサイト： http://www.adlinktech.com

販売およびサービス： Service@adlinktech.com

TEL: +886-2-82265877

FAX: +886-2-82265717

住所： 9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan

迅速で納得できるサービスを得るため、以下のサービス用紙の必要事項をすべて記入してメールまたは FAX で送信してください。

会社情報	
会社 / 組織名	
担当者氏名	
E メールアドレス	
アドレス	
国名	
TEL	FAX:
ウェブサイト	
製品情報	
製品モデル名	
環境	OS: マザーボード： CPU: チップセット： BIOS:

問題を詳しく説明してください：

目次

表目次

図目次

1 製品紹介

PCI-8158 は PCI インタフェースを搭載した高機能高密度の 8 軸モーション・コントローラ・カードで、高周波パルス（6.55MHz）を発生してステッパまたはサーボ・モータを駆動できます。また、モーション・コントローラとして、8 軸直線および円形補間、速度が連続した連続補間を提供できます。1 軸動作ではオン・ザ・フライによる速度 / 位置の変更も可能です。

1 つのシステムで複数の PCI-8158 カードが使用できます。全 8 軸のインクリメンタル・エンコーダ・インタフェースには、不正確な機械的転送に起因する位置エラーを訂正する機能が備えられています。

PCI-8158 は全く新しいデザインを採用しています。キャリア・ボードは 8 軸パルス・トレイン出力制御チャンネルを搭載しています。高速トリガや分散 I/O 制御といった機能が必要な場合はドータ・ボードを追加できます。PCI-8158 は位置比較機能をサポートしています。直線スキャン・アプリケーションでは、モーション・コントローラは高速のトリガ・パルスを発生して、高解像度画像を取得する必要があります。この場合、DB-8150 を追加すると、PCI-8158 の機能を拡張できます。マシン・オートメーションではモーション・コントロール設計だけでなく、センサーや作動装置もカギとなります。通常、コントローラのセンサーや作動装置を統合するには I/O が必要です。ADLINK はそれらのデバイスを接続する別の方法、すなわち分散 I/O も提供しています。ドータ・カードを使用すれば、PCI-8158 に分散 I/O 機能を追加できます。この構成は配線の手間を軽減し、コントローラの実際のサイズを縮小できるだけでなく、費用対効果にも優れています。

図 1-1 は PCI-8158 カードの機能ブロック図です。モーション・コントロール機能には、台形および S 曲線加速 / 減速、2 軸間の直線および円形補間、連続モーション位置決め、13 の原点復帰モードが含まれています。上記の機能と複雑な計算はすべて ASIC 内部で実行されるので、CPU の負荷は軽減されます。

PCI-8158 は以下の 3 つのユーザー・フレンドリーな機能もサポートしています。

1. カード・インデックス設定 :

   PCI-8158 は DIP スイッチ設定でカードのインデックスを割り当てることができます。値は 0 から 15 までです。全体の制御

システムが巨大な場合、マシン・メーカーにとってカードのインデックスを判別できるのは役立ちます。

2. 緊急入力

緊急入力ピンを使うと、緊急事態発生時、パルス出力の送信を停止するようボードに促す緊急停止ボタンを配置できます。

3. ソフトウェアのセキュリティ保護

セキュリティ保護設計では、EEPROM に 16 ビット値を設定できます。ユーザーのインタフェース・プログラムはこの EEPROM を使って、不正はユーザーからソフトウェアおよびハードウェアを保護できます。

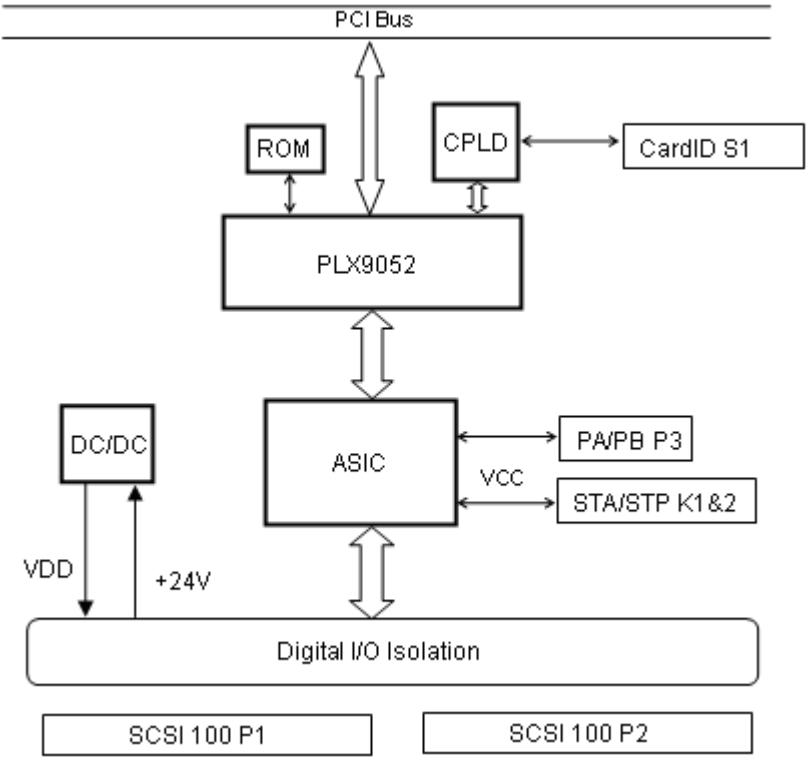


Figure 1-1: PCI-8158 のブロック図

MotionCreatorPro は PCI-8158 に同梱される Windows ベースのアプリケーション開発ソフトウェア・パッケージです。*MotionCreatorPro* はプロジェクト設計時のモーション・コントロール・システムのデバッグに役立ちます。オンスクリーン・ディスプレイには、インストールされているすべての軸の情報と PCI-8158 の I/O 信号ステータスが表示されます。

C++ コンパイラおよび Visual Basic 用に Windows プログラミング・ライブラリも提供されています。また、機能の動作を示すサンプル・プログラムも用意されています。

図 1-2 はアプリケーションの開発に本書を使用した推奨手順のフロー・チャートです。各手順については該当する章を参照してください。

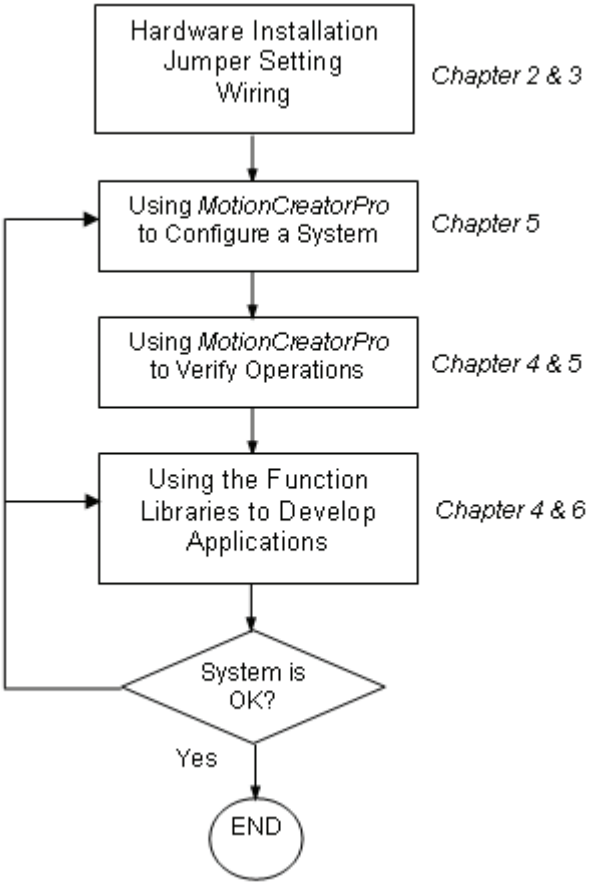


Figure 1-2: アプリケーション構築のフロー・チャート

1.1 特徴

以下のリストは PCI-8158 モーション・コントロール・システムの主な特徴の要約です。

- 32-bit PCI バス Plug & Play 対応（ユニバーサル）
- ステッピングまたはサーボ・モータ制御用 8 軸ステップおよび方向パルス出力
- 最大出力周波数 : 6.55 MPPS
- パルス出力オプション : OUT/DIR, CW/CCW
- 全モードで加速および減速時間がプログラム可能
- 全モードで台形または S 曲線速度プロファイルをサポート
- 2 ～ 4 軸直線補間
- 2 軸円形補間
- モーション後の輪郭連続補間
- ポジションおよび速度をオン・ザ・フライで変更
- 13 の自動検索対応原点復帰モード
- ハードウェア・バックラッシュ補償器および振動抑制
- 各軸で 2 ソフトウェア・エンド・リミットをサポート
- インクリメンタル・エンコーダ・フィードバック用 28-bit アップ / ダウン・カウンタ
- 全軸に原点スイッチ、インデックス信号（EZ）、正および負のエンド・リミット・スイッチ・インタフェースをサポート
- 8 軸高速位置ラッチ入力
- 8 軸位置比較およびトリガ出力（高速ではない。高速トリガ出力機能を追加するには DB-8150 の購入が必要）
- すべてのデジタル入力および出力信号が 2500Vrms 絶縁
- プログラマブル割り込みソース
- 複数軸でモーションの同時スタート / ストップ
- 手動パルス入力インタフェース
- カード・インデックス選択
- EERPOM によるセキュリティ保護
- 専用緊急入力ピンへの配線が可能

- ソフトウェアは 1 つのシステムで最大12のPCI-8158カード動作をサポート
- コンパクトな PCB 設計
- Microsoft WindowsI ベースのアプリケーション開発ソフトウェア MotionCreatorPro 同梱
- Windows 2000/XP 用 PCI-8158 ライブラリおよびユーティリティをサポート

1.2 仕様

- 使用可能モータ :
  - ステッピング・モータ
  - パルス・トレイン入力サーボ・ドライバによる AC または DC サーボ・モータ
- 性能 :
  - コントロール可能な軸数 : 8
  - 最大パルス出力周波数 : 6.55MPPS、直線、台形、またはS 曲線速度プロファイル駆動
  - 内部参照クロック : 19.66 MHz
  - 28-bitアップ/ダウン・カウンタ範囲: 0～268,435,455または −134,217,728 ～ +134,217,727
  - 位置パルス設定範囲（28-bit）: −134,217,728 ～ +134,217,728
  - パルス・レート設定範囲（パルス比 = 1: 65535）:

    0.1 PPS ～ 6553.5 PPS（マルチプライア = 0.1）

    1 PPS ～ 65535 PPS（マルチプライア = 1）

100 PPS ～ 6553500 PPS（マルチプライア = 100）

- I/O 信号 :
  - 軸ごとに入力 / 出力信号をサポート
  - すべての I/O 信号が 2500Vrms 絶縁電圧で光絶縁
  - コマンド・パルス出力ピン : OUT および DIR
  - インクリメンタル・エンコーダ信号入力ピン : EA および EB
  - エンコーダ・インデックス信号入力ピン : EZ
  - 機械式リミット / 原点信号入力ピン : □}EL、ORG
  - コンポジット・ピン : DI/LTC（ラッチ）/SD（減速）/ PCS（位置変更信号）/CLR（クリア）/EMG（緊急入力）
  - サーボ・モータ・インタフェース I/O ピン : INP、ALM、ERC
  - 汎用デジタル出力ピン : SVON、DO
  - 汎用デジタル入力ピン : RDY、GDI
  - パルス信号入力ピン : PA および PB（絶縁対応）
  - 同時スタート / ストップ信号 : STA および STP
- 共通仕様
  - コネクタ : 68 ピン SCSI タイプ・コネクタ
  - 稼働時温度 : 0□~C – 50□~C
  - 非稼働時温度 : –20□~C – 80□~C
  - 湿度 : 5 ～ 85%（結露なきこと）
- 電力消費
  - スロット電源（入力）: +5V DC□}5%、最大 900mA
  - 外部電源（入力）: +24V DC□}5%、最大 500mA
  - 外部電源（出力）: +5V DC□}5%、最大 500mA
- PCI-8158 寸法（PCB サイズ）: 185mm (L) X 100 mm (W)

1.3 対応ソフトウェア

1.3.1 プログラミング・ライブラリ

PCI-8158 ユーザーには Windows 2000/XP DLL が提供されています。上記関数ライブラリはボードに同梱されています。

1.3.2 MotionCreatorPro

この Windows ベースのユーティリティはカード、モータ、システムのセットアップに使用されます。また、ハードウェアやソフトウェアの問題解決にも役立ちます。同ユーティリティを使えば、I/O 論理パラメータがユーザーのプログラムにロードされるように設定できます。この製品もカードに同梱されます。

詳しくは 5 章を参照してください。

1.4 使用可能な端子ボード

ADLINK では接続に便利なサーボおよびステッパ用端子ボードを用意しています。ステッパではピン -to- ピン端子ボードの DIN-100S を使用し、サーボでは DIN-814M、DIN-814M-J3A、DIN-814Y、DIN-814P-A4 を使用します。対応するサーボは下表の通りです :

Mitsubishi J2 Super	DIN-814M
Mitsubishi J3A	DIN-814M-J3A
Yaskawa Sigma II	DIN-814Y
Panasonic MINAS A4	DIN-814P-A4

Table 1-1: 使用可能な端子ボード

2 インストール

本章では PCI-8158 の取り付け方法を説明します。以下の手順に従ってください:

- パッケージの確認（セクション 2.1）
- PCB の確認（セクション 2.2）
- ハードウェアの取り付け（セクション 2.3）
- ソフトウェア・ドライバのインストール（セクション 2.4）
- I/O 信号接続（3 章）とその動作（4 章）の理解
- コネクタのピン・アサインの理解と配線（その他のセクション）

2.1 同梱物

パッケージにはこのユーザーガイドのほかに、以下のアイテムも含まれています:

- PCI-8158: 高機能 8 軸サーボ / ステッパ・モーション・コントロール・カード
- ADLINK オールインワン CD

端子ボードは別売のアクセサリなので、PCI-8158 のパッケージには同梱されません。

不足しているものや損傷しているものがあれば、製品を購入したディーラーにお知らせください。製品を送付したり、保管したりする場合に備えて、梱包材料やカートンは保存しておいてください。

2.2 PCI-8158 の輪郭図

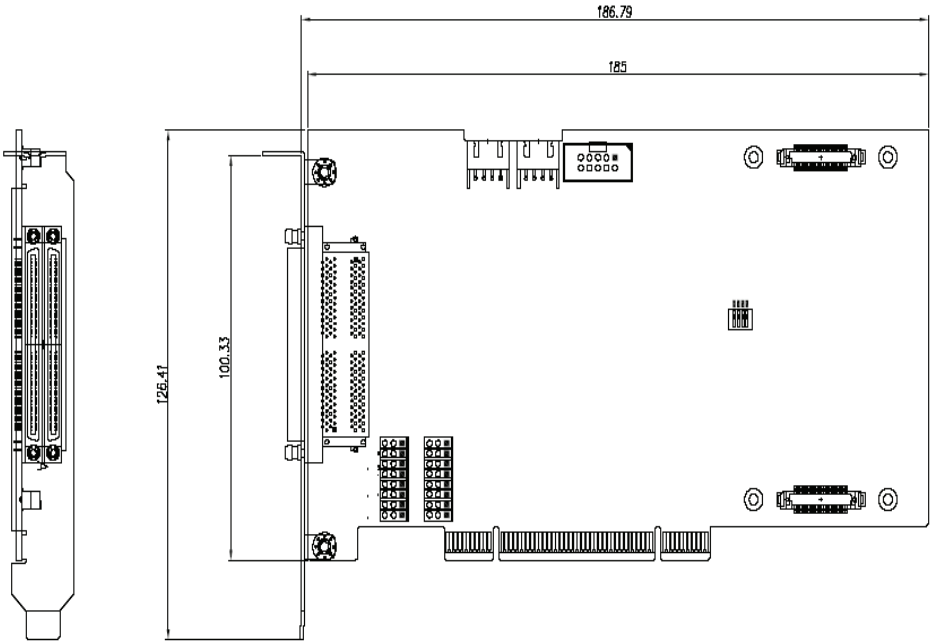


Figure 2-1: PCI-8158 の PCB レイアウト

- P1/P2: 入力 / 出力信号コネクタ（100 ピン）
- K1/K2: 同時スタート / ストップ・コネクタ
- P3: 手動パルサー
- S1: カード・インデックス選択用 DIP スイッチ（0-15）
- J1-J16: パルス出力選択ジャンパ（回線ドライバ / オープン・コレクタ）

2.3 PCI-8158 ハードウェアの取り付け

2.3.1 ハードウェア設定

PCI-8158 は Plug & Play にフル対応です。したがって、メモリ割り当て（I/O ポート・ロケーション）と PCI カードの IRQ チャンネルはシステム BIOS によって割り当てられます。アドレスの割り当

てはシステムのすべての PCI カードに対してボードごとに行われます。

2.3.2 PCI スロットの選択

コンピュータ・システムが採用するスロットには PCI と ISA の両方があります。PCI カードを PC/AT スロットに無理に挿入しないでください。PCI-8158 は PCI スロットのみで使用できます。

2.3.3 取り付け手順

1. 本書を最後まで読み、必要に応じてジャンパを設定します。
2. コンピュータの電源を切ります。コンピュータに接続されているすべてのアクセサリ（プリンタ、モデム、モニタなど）の電源を切ります。コンピュータのカバーを外します。
3. 取り付け可能な 32-bit PCI 拡張スロットを選択します。PCI スロットは ISA や EISA スロットよりも短く、通常白または象牙色をしています。
4. PCI-8158 を取り付ける前に、コンピュータの金属ケースに触れるなどして体に蓄積している静電気を放電させます。カードの端を持ち、コンポーネントには手を触らないでください。
5. 選択した PCI スロットの位置にカードを合わせます。
6. スロットから外したネジを使って、システムのリア・パネルの所定の場所にカードを固定します。

2.3.4 トラブルシューティング

システムが再起動しない場合、または PCI ボードが正しく機能しない場合、（不正な ISA 設定などのため）割り込みが衝突しているのかもしれません。単純なミスによるのではない場合は、システムに同梱されているBIOSの説明書を参照して問題を解決してください。

Windows システムのコントロールパネルからカードがシステムに表示されているかどうか確認します。システムに表示されていない場合は、BIOS の PCI 設定を確認するか、他の PCI スロットを使用してください。

2.4 ソフトウェア・ドライバのインストール

1. ADLINK オールインワン CD を自動実行します。「Driver Installation -> Motion Control -> PCI-8158」を選択します。
2. インストーラの手順に従います。
3. セットアップ・インストール完了後、Windows を再起動します。

注意： 必要な場合は、ADLINK のウェブサイトから最新のソフトウェアをダウンロードしてください。

2.5 P1/P2 ピン・アサイン : 主コネクタ

P1/P2 はモーション・コントロール I/O 信号の主コネクタです。

番号	名称	I/O	関数	番号	名称	I/O	関数
1	VDD	O	+5V 電源出力	51	VDD	O	+5V 電源出力
2	EXGND	–	外部電源グラウンド	52	EXGND	–	外部電源グラウンド
3	OUT0+	O	パルス信号（+）	53	OUT2+	O	パルス信号（+）
4	OUT0–	O	パルス信号（–）	54	OUT2–	O	パルス信号（–）
5	DIR0+	O	方向信号（+）	55	DIR2+	O	方向信号（+）
6	DIR0–	O	方向信号（–）	56	DIR2–	O	方向信号（–）
7	SVON0	O	サーボ・オン / オフ	57	SVON2	O	サーボ・オン / オフ
8	ERC0	O	偏差カウンタ・クリア Signal	58	ERC2	O	偏差カウンタ・クリア信号
9	ALM0	I	アラーム信号	59	ALM2	I	アラーム信号
10	INP0	I	インポジション信号	60	INP2	I	インポジション信号
11	RDY0	I	多目的入力信号	61	RDY2	I	多目的入力信号
12	EXGND		外部電源グラウンド	62	EXGND		外部電源グラウンド
13	EA0+	I	エンコーダ A 相（+）	63	EA2+	I	エンコーダ A 相（+）
14	EA0–	I	エンコーダ A 相（–）	64	EA2–	I	エンコーダ A 相（–）
15	EB0+	I	エンコーダ B 相（+）	65	EB2+	I	エンコーダ B 相（+）
16	EB0–	I	エンコーダ B 相（–）	66	EB2–	I	エンコーダ B 相（–）
17	EZ0+	I	エンコーダ Z 相（+）	67	EZ2+	I	エンコーダ Z 相（+）
18	EZ0–	I	エンコーダ Z 相（–）	68	EZ2–	I	エンコーダ Z 相（–）
19	VDD	O	+5V 電源出力	69	VDD	O	+5V 電源出力
20	EXGND	–	外部電源グラウンド	70	EXGND	–	外部電源グラウンド
21	OUT1+	O	パルス信号（+）	71	OUT3+	O	パルス信号（+）
22	OUT1–	O	パルス信号（–）	72	OUT3–	O	パルス信号（–）
23	DIR1+	O	方向信号（+）	73	DIR3+	O	方向信号（+）
24	DIR1–	O	方向信号（–）	74	DIR3–	O	方向信号（–）
25	SVON1	O	サーボ・オン / オフ	75	SVON3	O	サーボ・オン / オフ
26	ERC1	O	偏差カウンタ・クリア信号	76	ERC3	O	偏差カウンタ・クリア信号
27	ALM1	I	アラーム信号	77	ALM3	I	アラーム信号
28	INP1	I	インポジション信号	78	INP3	I	インポジション信号
29	RDY1	I	多目的入力信号	79	RDY3	I	多目的入力信号
30	EXGND		外部電源グラウンド	80	EXGND		外部電源グラウンド
31	EA1+	I	エンコーダ A 相（+）	81	EA3+	I	エンコーダ A 相（+）
32	EA1–	I	エンコーダ A 相（–）	82	EA3–	I	エンコーダ A 相（–）
33	EB1+	I	エンコーダ B 相（+）	83	EB3+	I	エンコーダ B 相（+）

Table 2-1: P1/P2 ピン・アサイン

番号	名称	I/O	関数	番号	名称	I/O	関数
34	EB1−	I	エンコーダ B 相（−）	84	EB3−	I	エンコーダ B 相（−）
35	EZ1+	I	エンコーダ Z 相（+）	85	EZ3+	I	エンコーダ Z 相（+）
36	EZ1−	I	エンコーダ Z 相（−）	86	EZ3−	I	エンコーダ Z 相（−）
37	PEL0	I	エンド・リミット信号（+）	87	PEL2	I	エンド・リミット信号（+）
38	MEL0	I	エンド・リミット信号（−）	88	MEL2	I	エンド・リミット信号（−）
39	GDI0	I	DI/LTC/PCS/SD/CLR0	89	GDI2	I	DI/LTC/PCS/SD/CLR2
40	DO0	O	汎用出力 0	90	DO2	O	汎用出力 2
41	ORG0	I	原点信号	91	ORG2	I	原点信号
42	EXGND		外部電源グラウンド	92	EXGND		外部電源グラウンド
43	PEL1	I	エンド・リミット信号（+）	93	PEL3	I	エンド・リミット信号（+）
44	MEL1	I	エンド・リミット信号（−）	94	MEL3	I	エンド・リミット信号（−）
45	GDI1	I	DI/LTC/PCS/SD/CLR1/ EMG	95	GDI3	I	DI/LTC/PCS/SD/CLR3
46	DO1	O	汎用出力 1	96	DO3	O	汎用出力 3
47	ORG1	I	原点信号	97	ORG3	I	原点信号
48	EXGND	−	外部電源グラウンド	98	EXGND	−	外部電源グラウンド
49	EXGND	−	外部電源グラウンド	99	E_24V	−	絶縁電源入力、+24V
50	EXGND	−	外部電源グラウンド	100	E_24V	−	絶縁電源入力、+24V

Table 2-1: P1/P2 ピン・アサイン

► P1 は軸 0 から 3 を制御し、P2 は軸 4 から 7 を制御します。

2.6 K1/K2 ピン・アサイン : 同時スタート / ストップ

K1 および K2 は複数軸または複数カードの同時スタート / ストップ信号を送信します。

番号	名称	関数
1	+5V	PCI バス電源出力（VCC）
2	STA	同時スタート信号入力 / 出力
3	STP	同時ストップ信号入力 / 出力
4	GND	PCI バス電源グラウンド

Table 2-2: K1/K2 ピン・アサイン

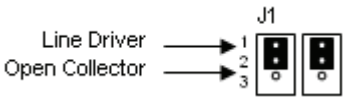
注意: +5VピンおよびGNDピンの電源はPCIバスから供給されます。

2.7　パルス出力用 J1 ～ J16 ジャンパの設定

J1 ～ J16 はパルス出力信号（DIR および OUT）の種類を設定するのに使用されます。出力信号の種類は差動回線ドライバかオープン・コレクタ出力のどちらかです。ジャンパ設定の詳細についてはセクション 3.1 を参照してください。デフォルト設定は差動回線ドライバ・モードです。対応表は以下の通りです :

JP1 および JP2	軸 0	JP9 および JP10	軸 4
JP3 および JP4	軸 1	JP11 および JP12	軸 5
JP5 および JP6	軸 2	JP13 および JP14	軸 6
JP7 および JP8	軸 3	JP15 および JP16	軸 7

Table 2-3: J1 から J16 までのジャンパ設定



2.8　カード・インデックス用 S1 スイッチ設定

S1 スイッチはカード・インデックスの設定に使用されます。例えば、1 をオンに、その他をオフにすると、カード・インデックスは 1 になります。値は 0 から 15 までです。詳しくは下表を参照してください。

カード ID	スイッチ設定（オン=1）
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Table 2-4: S1 スイッチ設定

2.9 P3 手動パルス

P3 の信号は手動パルス入力に使用されます。

番号	名称	機能（軸）
1	VDD	絶縁電源、+5V
2	PA+	パルス A+ 相信号入力
3	PA−	パルス A− 相信号入力
4	PB+	パルス B+ 相信号入力
5	PB−	パルス B− 相信号入力
6	EXGND	外部グラウンド
7	なし	未使用
8	なし	未使用
9	なし	未使用

Table 2-5: P3 手動パルス

注意： +5V ピンおよび GND ピンの電源は PCI バスから直接供給されます。したがって、それらの信号は絶縁されていません。

3 信号接続

本章ではすべての I/O の信号接続について説明します。PCI-8158 とモータ・ドライバをケーブルで接続する前に、本章の内容を参照してください。

本章には以下のセクションが含まれます :

3.1 パルス出力信号 OUT および DIR

PCI-8158 には 8 軸のパルス出力信号があります。各軸では、OUT および DIR の差動信号からなる 2 つのペアがパルス・トレインを送信し、方向を示すのに使用されます。OUT 信号と DIR 信号は CW 信号と CCW 信号のペアとしてもプログラムできます。OUT 信号と DIR 信号の詳しい論理特性についてはセクション 4.1.1 を参照してください。同セクションでは、OUT 信号と DIR 信号の電気的特性が詳しく説明されています。各信号は 1 組の差動信号から構成されます。例えば、OUT0 は OUT0+ 信号と OUT0- 信号から構成されます。P1 のすべてのパルス出力信号は下表の通りです。

P1 のピン番号	信号名	説明	軸 #
3	OUT0+	パルス信号（+）	0
4	OUT0−	パルス信号（−）	0
5	DIR0+	方向信号（+）	0
6	DIR0−	方向信号（−）	0
21	OUT1+	パルス信号（+）	1
22	OUT1−	パルス信号（−）	1
23	DIR1+	方向信号（+）	1
24	DIR1−	方向信号（−）	1
53	OUT2+	パルス信号（+）	2
54	OUT2−	パルス信号（−）	2
55	DIR2+	方向信号（+）	2
56	DIR2−	方向信号（−）	2
71	OUT3+	パルス信号（+）	3
72	OUT3−	パルス信号（−）	3
73	DIR3+	方向信号（+）	3
74	DIR3−	方向信号（−）	3

Table 3-1: パルス出力信号 OUT（P1）

P2 のピン番号	信号名	説明	軸 #
3	OUT4+	パルス信号（+）	4
4	OUT4−	パルス信号（−）	4
5	DIR4+	方向信号（+）	4
6	DIR4−	方向信号（−）	4
21	OUT5+	パルス信号（+）	5
22	OUT5−	パルス信号（−）	5
23	DIR5+	方向信号（+）	5
24	DIR5−	方向信号（−）	5
53	OUT6+	パルス信号（+）	6
54	OUT6−	パルス信号（−）	6
55	DIR6+	方向信号（+）	6
56	DIR6−	方向信号（−）	6
71	OUT7+	パルス信号（+）	7
72	OUT7−	パルス信号（−）	7
73	DIR7+	方向信号（+）	7
74	DIR7−	方向信号（−）	7

Table 3-2: パルス出力信号 OUT（P2）

OUT 信号または DIR 信号の出力はジャンパを使って、差動回線ドライバまたはオープン・コレクタ出力として設定できます。ジャンパ J1 から J16 の 1 と 2 または 2 と 3 の間のジャンパ回線によって出力モードを以下のように選択できます。

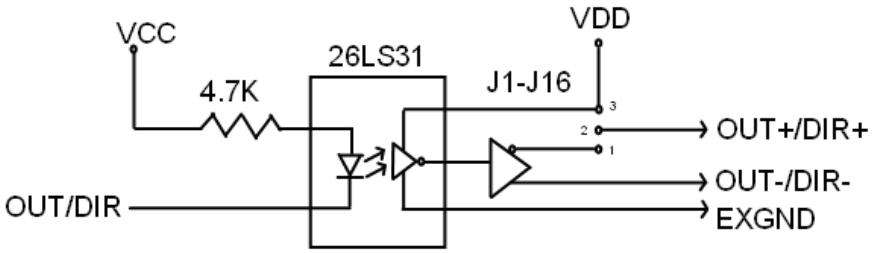
出力信号	差動回線ドライバ出力では、以下の 1 と 2 間の切断を閉じます：	オープン・コレクタ出力では、以下の 2 と 3 間の切断を閉じます：
OUT0+	J1	J1
DIR0+	J9	J9
OUT1+	J2	J2
DIR1+	J10	J10
OUT2+	J3	J3
DIR2+	J11	J11
OUT3+	J4	J4

出力信号	差動回線ドライバ出力では、以下の 1 と 2 間の切断を閉じます :	オープン・コレクタ出力では、以下の 2 と 3 間の切断を閉じます :
DIR3+	J12	J12
OUT4+	J5	J5
DIR4+	J13	J13
OUT5+	J6	J6
DIR5+	J14	J14
OUT6+	J7	J7
DIR6+	J15	J15
OUT7+	J8	J8
DIR7+	J16	J16

Table 3-3: 出力信号

OUT と DIR のデフォルト設定は差動回線ドライバ・モードに設定されています。

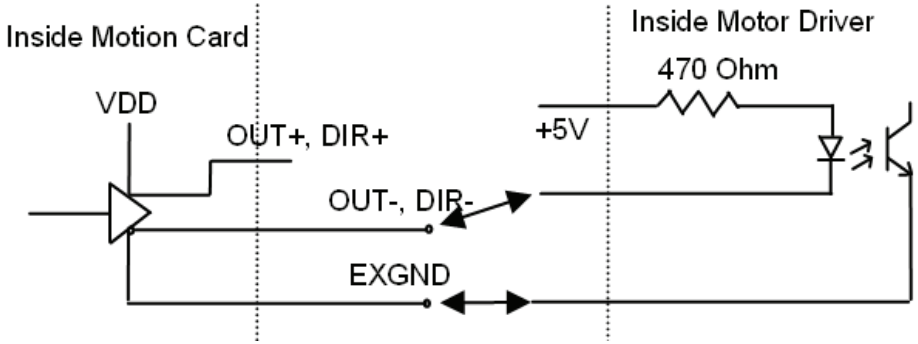
2 つの軸の OUT 信号と DIR 信号の配線図は以下の通りです。



íçà": パルス出力がオープン・コレクタ出力モードに設定されている場合、OUT- と DIR- が OUT 信号と DIR 信号を送信するのに使用されます。**OUT-ÉsÉìÇýDIR-ÉsÉìÇÃÉVÉìÉNìdó¨ÇÕ20mAà»â½ÇÝÇ»ÇØÇÍÇŒÇ»ÇËÇ‹Ç¼ÇÒÅB** デフォルト設定では 1-2 がショートされています。

提案される使用方法 : 下図のように、ジャンパ 2-3 をショートし、OUT-/DIR- をドライバの 470Ω パルス入力インタフェースの COM

に接続します。ドライバの OUT/DIR に接続する OUT−/DIR− を選択します。



警告 : シンク電流は 20mA 以下でなければなりません。20mA を超えると、26LS31 の故障の原因となります。

3.2 エンコーダ・フィードバック信号 EA、EB、EZ

エンコーダ・フィードバック信号には、EA、EB、EZ があります。それぞれの軸には、相 A（EA）入力、相 B（EB）入力、インデックス（EZ）入力の 3 つの作動ペアからなる 6 つのピンがあります。EA と EB は位置カウントに使用され、EZ はゼロ位置のインデックスに使用されます。信号名、ピン番号、軸番号の対応は下表の通りです :

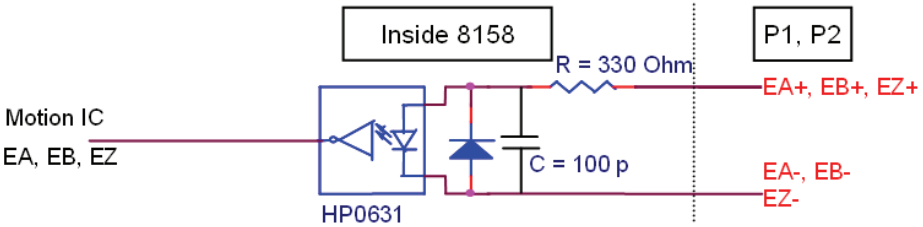
P1 のピン番号	信号名	軸 #	P1 のピン番号	信号名	軸 #
13	EA0+	0	14	EA0−	0
15	EB0+	0	16	EB0−	0
31	EA1+	1	32	EA1−	1
33	EB1+	1	34	EB1−	1
63	EA2+	2	64	EA2−	2
65	EB2+	2	66	EB2−	2
81	EA3+	3	82	EA3−	3
83	EB3+	3	84	EB3−	3

P2 のピン番号	信号名	軸 #	P2 のピン番号	信号名	軸 #
13	EA4+	4	14	EA4−	4
15	EB4+	4	16	EB4−	4
31	EA5+	5	32	EA5−	5
33	EB5+	5	34	EB5−	5
63	EA6+	6	64	EA6−	6
65	EB6+	6	66	EB6−	6
81	EA7+	7	82	EA7−	7
83	EB7+	7	84	EB7−	7

P1 のピン番号	信号名	軸 #	P1 のピン番号	信号名	軸 #
17	EZ0+	0	18	EZ0−	0
35	EZ1+	1	36	EZ1−	1
67	EZ2+	2	68	EZ2−	2
85	EZ3+	3	86	EZ3−	3

P2 のピン番号	信号名	軸 #	P2 のピン番号	信号名	軸 #
17	EZ4+	4	18	EZ4−	4
35	EZ5+	5	36	EZ5−	5
67	EZ6+	6	68	EZ6−	6
85	EZ7+	7	86	EZ7−	7

EA 信号、EB 信号、EZ 信号の入力回路は以下の通りです：

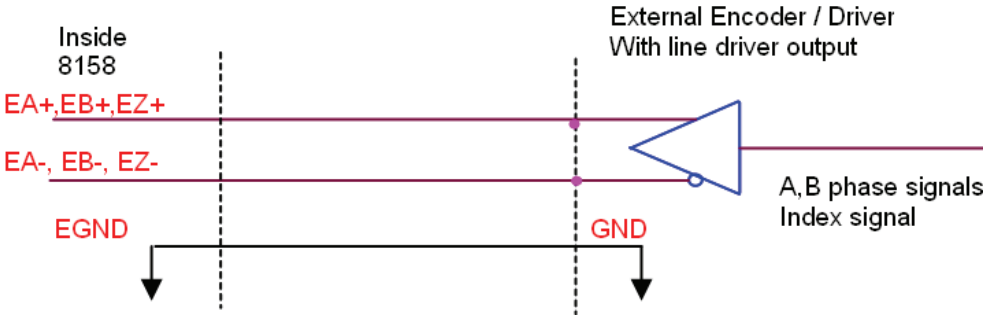


エンコーダ入力信号（EA+、EA−）、（EB+、EB−）、（EZ+、EZ−）のそれぞれの差動ペア間の電圧は 3.5V 以上でなければならないことに注意してください。したがって、エンコーダのフィードバックまたはモータ・ドライバのフィードバック接続時、出力電流がソースを超えないように注意しなければなりません。差動信号のペアはデジタル信号の EA、EB、EZ に変換されてから、モーション・コントロール ASIC に提供されます。

下図は入力信号と外部回路の接続の例です。入力回路は（1）差動回線ドライバ、または（2）オープン・コレクタ出力が備わっている場合、エンコードまたはモータ・ドライバに接続できます。

3.2.1 回線ドライバ出力の接続

PCI-8158 のエンコーダ入力を駆動するには、ドライバ出力が 8mA 以上の駆動容量を備え、差動ペア間に 3.5V 以上の電圧を提供しなければなりません。また、両端のグラウンドは結合されていなければなりません。最大周波数は配線距離および信号状況により 4MHz またはそれ以上になります。

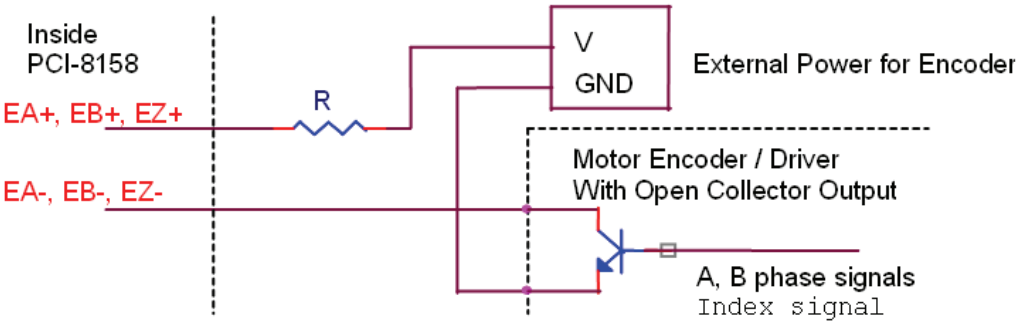


3.2.2　オープン・コレクタ出力の接続

オープン・コレクタ出力を接続するには、外部電源が必要です。モータ・ドライバの中には、電源を備えているものもあります。PCI-8158、エンコーダ、電源の接続は下図の通りです。PCI-8158 の入力回路を保護するために、外部電流を抑制する抵抗 R が必要であることに注意してください。エンコーダの電源により提案されている抵抗値は下表の通りです。

エンコーダ電圧 (V)	外部抵抗 R
+5V	0Ω(なし)
+12V	1.5kΩ
+24V	3.0kΩ

I_f = 8mA



エンコーダのフィードバック信号の詳しい動作情報については、セクション 4.4 を参照してください。

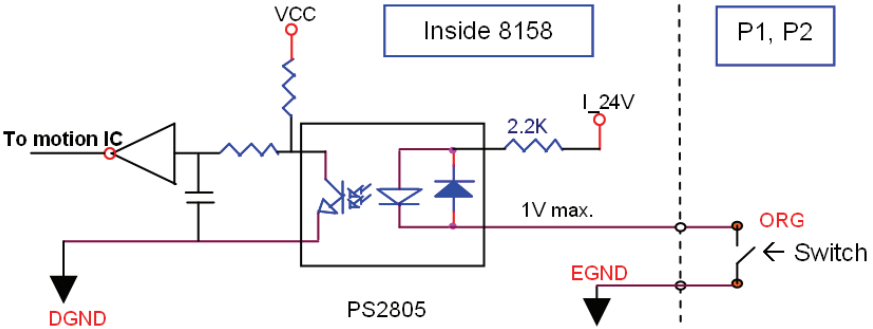
3.3 原点信号 ORG

原点信号（ORG0 ～ ORG7）はメカニズムの原点の入力信号として使用されます。信号名、ピン番号、軸番号は下表の通りです：

P1 のピン番号	信号名	軸 #
41	ORG0	0
47	ORG1	1
91	ORG2	2
97	ORG3	3

P2 のピン番号	信号名	軸 #
41	ORG4	4
47	ORG5	5
91	ORG6	6
97	ORG7	7

ORG 信号の入力回路は下図の通りです。通常、軸の原点を示すのにリミット・スイッチが使用されます。リミット・スイッチの仕様は +24V @ 6mA 以上の接点容量がなければなりません。不具合の原因となる高周波数スパイクを排除するために、内部フィルタ回路が使用されます。



モーション・コントローラが原点復帰モードで動作している場合、ORG 信号は制御出力信号（OUT および DIR）を禁止するために使用されます。ORG 信号の詳しい動作については、セクション 4.3.3 を参照してください。

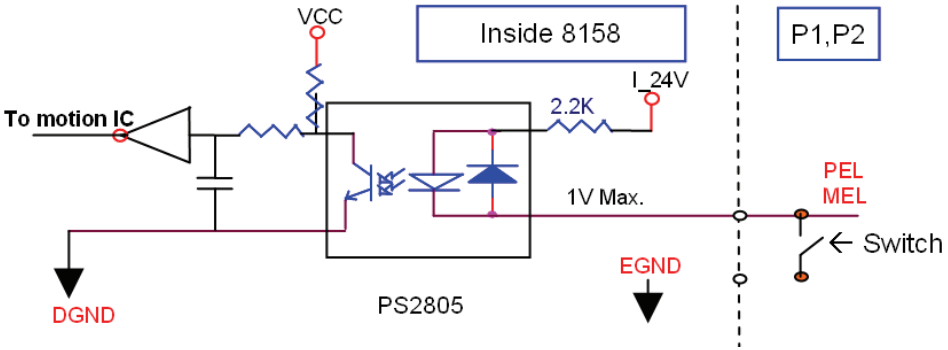
3.4 エンド・リミット信号 PEL および MEL

各軸には PEL と MEL の 2 つのエンド・リミット信号があります。PEL はエンド・リミット信号がプラス方向であることを示し、MEL はエンド・リミット信号がマイナス方向であることを示します。信号名、ピン番号、軸番号は下表の通りです :

P1 のピン番号	信号名	軸 #	P1 のピン番号	信号名	軸 #
37	PEL0	0	38	MEL0	0
43	PEL1	1	44	MEL1	1
87	PEL2	2	88	MEL2	2
93	PEL3	3	94	MEL3	3

P2 のピン番号	信号名	軸 #	P2 のピン番号	信号名	軸 #
37	PEL4	4	38	MEL4	4
43	PEL5	5	44	MEL5	5
87	PEL6	6	88	MEL6	6
93	PEL7	7	94	MEL7	7

回路図は下図の通りです。外部リミット・スイッチは +24V @ 8mA 以上の接点容量を備えていなければなりません。『A タイプ』（常時開）接点または『B タイプ』（常時閉）接点のいずれかのスイッチが使用できます。外部リミット信号のアクティブ論理を設定するには、_8158_set_limit_logic 関数の説明を参照してください。



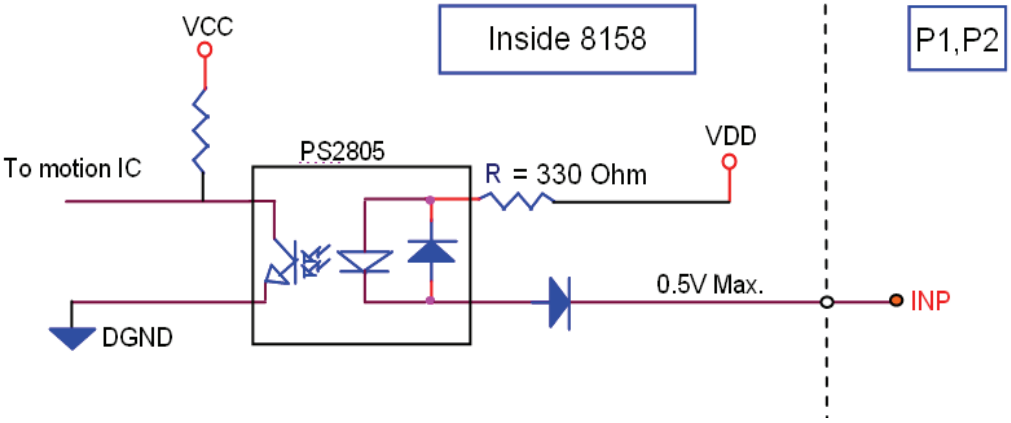
3.5　インポジション信号 INP

サーボ・モータ・ドライバからのインポジション信号 INP は偏差エラーを示します。偏差エラーが発生すると、サーボの位置はゼロになります。信号名、ピン番号、軸番号は下表の通りです :

P1 のピン番号	信号名	軸 #
10	INP0	0
28	INP1	1
60	INP2	2
78	INP3	3

P2 のピン番号	信号名	軸 #
10	INP4	4
28	INP5	5
60	INP6	6
78	INP7	7

INP 信号の入力回路は下図に通りです :



通常、インポジション信号はサーボ・モータから発生されて、オープン・コレクタ出力信号になります。外部回路は INP 信号を駆動するために 8mA 以上の電流シンク容量を備えていなければなりません。

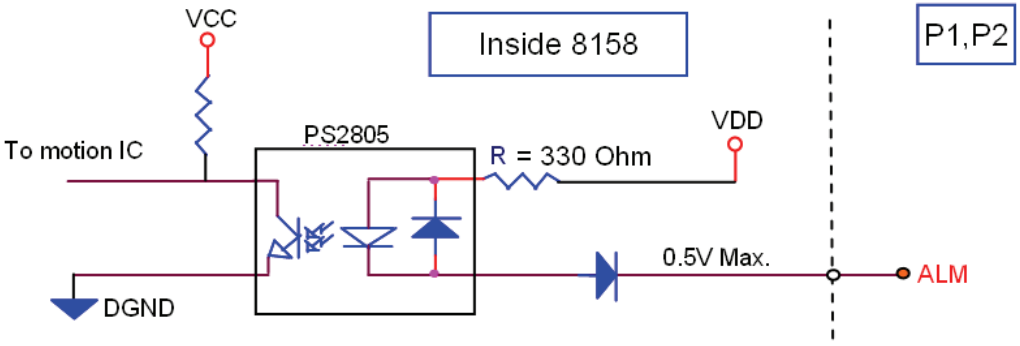
3.6 アラーム信号 ALM

アラーム信号 ALM はサーボ・ドライバからのアラーム状態を表示するのに使用されます。信号名、ピン番号、軸番号は下表の通りです :

P1 のピン番号	信号名	軸 #
9	ALM0	0
27	ALM1	1
59	ALM2	2
77	ALM3	3

P2 のピン番号	信号名	軸 #
9	ALM4	4
27	ALM5	5
59	ALM6	6
77	ALM7	7

入力アラーム回路は下図の通りです。通常、ALM 信号はサーボ・モータ・ドライバから発生されて、オープン・コレクタ出力信号になります。外部回路は ALM 信号を駆動するために 8mA 以上の電流シンク容量を備えていなければなりません。



3.7 偏差カウンタ・クリア信号 ERC

偏差カウンタ・クリア信号（ERC）は以下の 4 つの状況でアクティブになります：

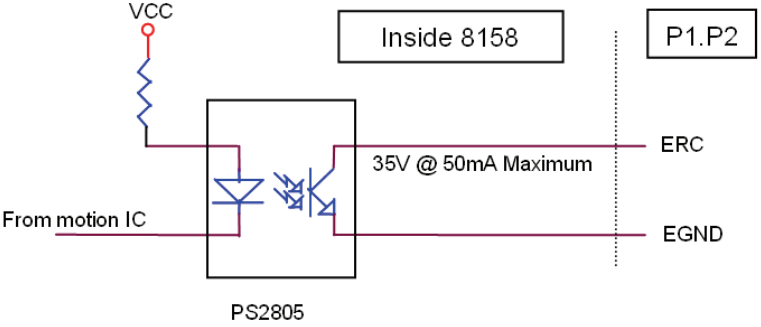
1. 原点復帰完了時
2. エンド・リミット・スイッチ・アクティブ時
3. アラーム信号が OUT および DIR 信号を停止した場合
4. ソフトウェア（オペレータ）から緊急停止コマンドが発行された場合

信号名、ピン番号、軸番号は下表の通りです：

P1 のピン番号	信号名	軸 #
8	ERC0	0
26	ERC1	1
58	ERC2	2
76	ERC3	3

P2 のピン番号	信号名	軸 #
8	ERC4	4
26	ERC5	5
58	ERC6	6
76	ERC7	7

ERC 信号はサーボ・モータ・ドライバの偏差カウンタをクリアするのに使用されます。ERC 出力回路は最大 35V @ 50mA の駆動容量を備えたオープン・コレクタです。



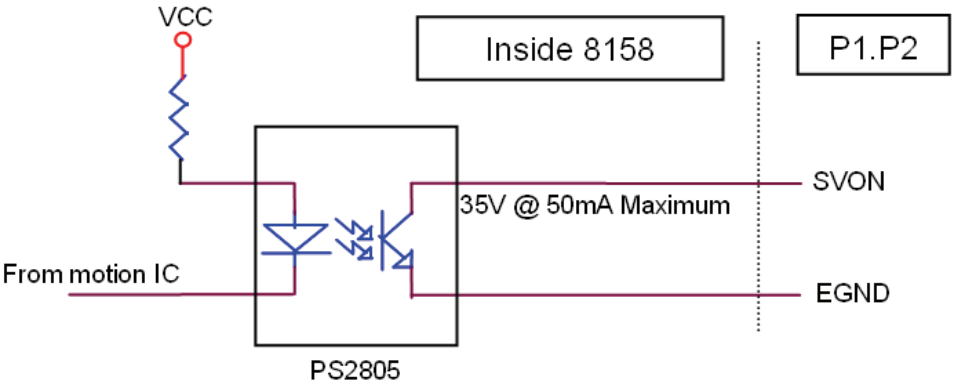
3.8 汎用信号 SVON

SVON 信号はサーボ・モータ制御または汎用出力信号として使用できます。信号名、ピン番号、軸番号は下表の通りです :

P1 のピン番号	信号名	軸 #
7	SVON0	0
25	SVON1	1
57	SVON2	2
75	SVON3	3

P2 のピン番号	信号名	軸 #
7	SVON4	4
25	SVON5	5
57	SVON6	6
75	SVON7	7

SVON 信号の出力回路は下図の通りです :



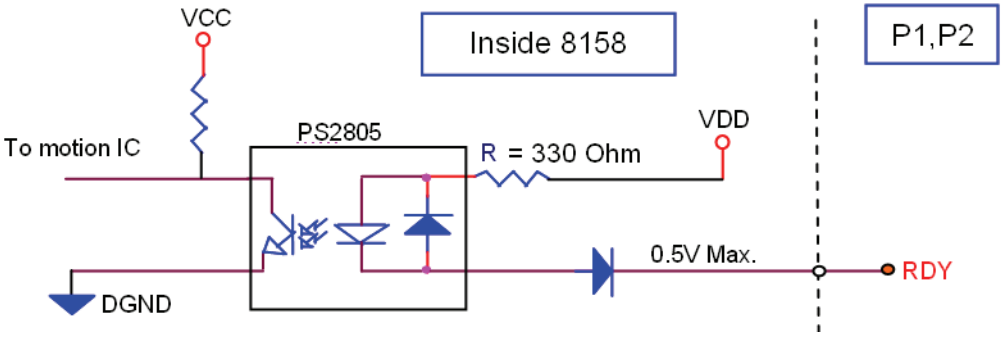
3.9 汎用信号 RDY

RDY 信号はモータ・ドライバ対応入力または汎用入力信号として使用できます。信号名、ピン番号、軸番号は下表の通りです :

P1 のピン番号	信号名	軸 #
11	RDY0	0
29	RDY1	1
61	RDY2	2
79	RDY3	3

P2 のピン番号	信号名	軸 #
11	RDY4	4
29	RDY5	5
61	RDY6	6
79	RDY7	7

RDY 信号の入力回路は下図に通りです :



3.10 多機能出力ピン : DO/CMP

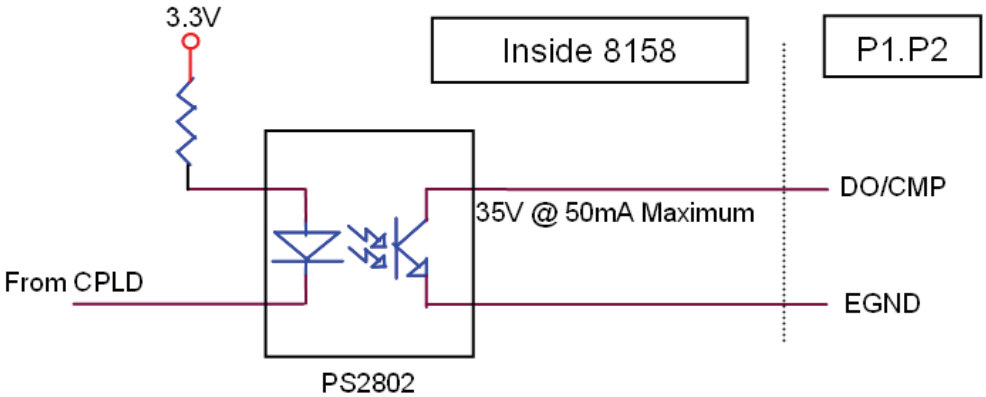
PCI-8158 は 8 つの軸に対応する DO/CMP0 から DO/CMP7 までの 8 つの多機能出力チャンネルを備えています。それぞれの出力ピンはデジタル出力（DO）または比較出力（CMP）として個別に設定できます。比較出力ピンに設定した場合、エンコーダ・カウンタがユーザー設定値に達すると、ピンはパルス出力を発生します。

多機能チャンネルは P1 および P2 に配置されます。信号名、ピン番号、軸番号は下表の通りです :

P1 のピン番号	信号名	軸 #
40	DO/CMP0	0
46	DO/CMP1	1
90	DO/CMP2	2
96	DO/CMP3	3

P2 のピン番号	信号名	軸 #
40	DO/CMP4	4
46	DO/CMP5	5
90	DO/CMP6	6
96	DO/CMP7	7

最初の 2 軸の CMP は以下の配線図の通りです :



3.11 多機能入力ピン : DI/LTC/SD/PCS/CLR/EMG

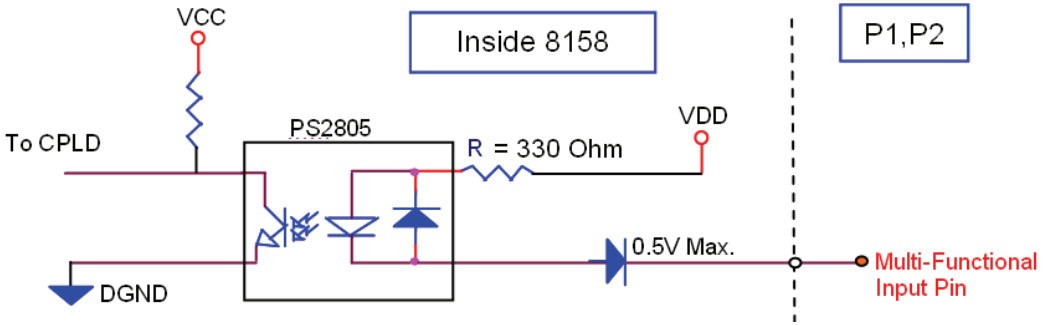
PCI-8158 は 8 つの多機能入力ピンを備えています。8 つのピンはそれぞれ DI（デジタル入力）、LTC（ラッチ）、SD（減速）、PCS（ターゲット位置のオーバーライド）、CLR（カウンタ・クリア）、EMG（緊急）のいずれかに設定できます。ピン機能の選択方法については、6.12 を参照してください。

多機能入力ピンは P1 と P2 です。信号名、ピン番号、軸番号は下表の通りです :

P1 のピン番号	信号名	軸 #
39	DI/LTC/SD/PCS/CLR/EMG_0	0
45	DI/LTC/SD/PCS/CLR/EMG_1	1
89	DI/LTC/SD/PCS/CLR/EMG_2	2
95	DI/LTC/SD/PCS/CLR/EMG_3	3

P2 のピン番号	信号名	軸 #
39	DI/LTC/SD/PCS/CLR/EMG_4	4
45	DI/LTC/SD/PCS/CLR/EMG_5	5
89	DI/LTC/SD/PCS/CLR/EMG_6	6
95	DI/LTC/SD/PCS/CLR/EMG_7	7

多機能入力ピンの配線図は以下の通りです :



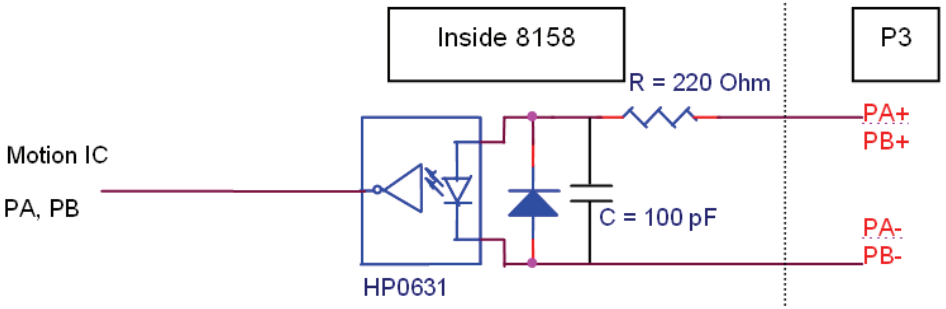
3.12 パルス入力信号 PA および PB（PCI-8158）

PCI-8158 は以下の PN1 のピンからの差動パルス入力信号を受信できます。パルスの動作はエンコーダと同様です。A-B 相信号はモータをガイドする位置決め情報を発行します。

P3 のピン番号	信号名	軸 #	P3 のピン番号	信号名	軸 #
2	PA+	0-7	3	PA-	0-7
4	PB+	0-7	5	PB-	0-7

パルス信号は軸0から軸7で使用されます。_8158_disable_pulser_input 関数を使用すれば、各軸でパルスを使用するか、使用しないを設定できます。

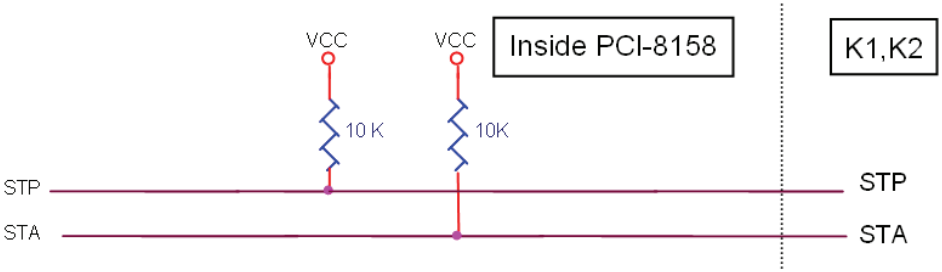
差動パルス入力ピンの配線図は以下の通りです：



3.13 同時スタート / ストップ信号 STA および STP

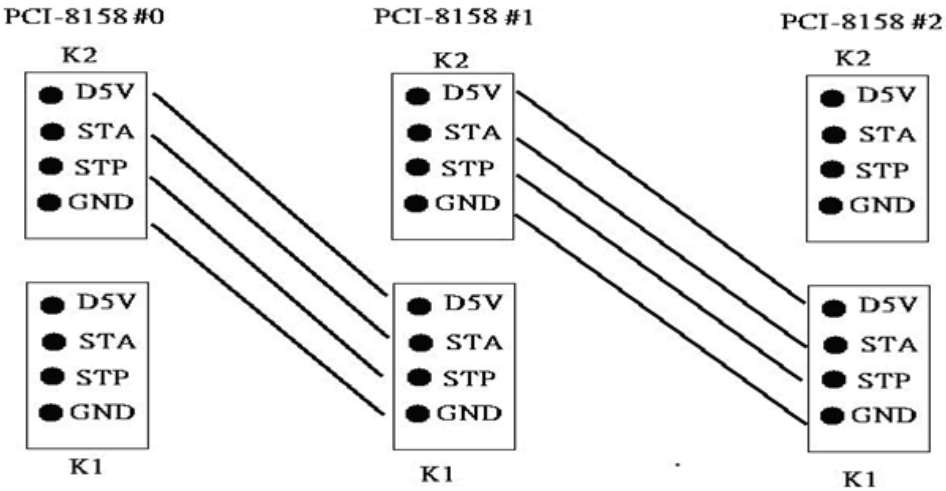
PCI-8158 は複数の軸のモーションの同時スタート / ストップを可能にする STA および STP 信号に対応しています。STA 信号と STP 信号は CN4 で使用されます。

オンボード回路は下図の通りです。4 つの軸の STA 信号と STP 信号はそれぞれ結合されています。

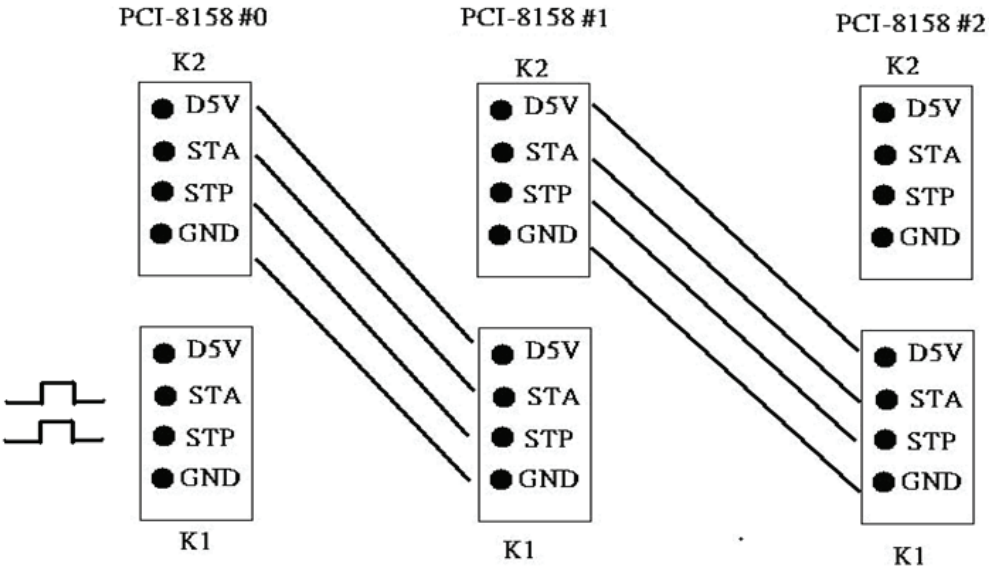


STA 信号および STP 信号は入力と出力の両方の信号です。スタート動作とストップ動作を同時に操作するには、ソフトウェア制御と外部制御の両方が必要です。ソフトウェア制御を使えば、どの PCI-8158 からでも信号を発生できるようになります。外部オープン・コレクタまたはスイッチを使えば、同時スタート / ストップ用 STA/ STP 信号を駆動できます。

PCI-8158 カードが複数ある場合、最初のカードの K2 コネクタと次のカードの K1 コネクタを接続してください。同じ PCI-8158 の K1 コネクタと K2 コネクタは内部で接続されています。



外部のスタートおよびストップ信号を使えば、複数のカードの同時モータ操作を発行することもできます。それは外部のスタートおよびストップ信号を最初のPCI-8158カードのK1コネクタのSTAピンと STP ピンに接続するだけで可能となります。


PCI-8158 #0
K2
D5V
STA
STP
GND
D5V
STA
STP
GND
K1
PCI-8158 #1
K2
D5V
STA
STP
GND
D5V
STA
STP
GND
K1
PCI-8158 #2
K2
D5V
STA
STP
GND
D5V
STA
STP
GND
K1

4 動作理論

この章ではモーション・コントローラ・カードの動作について詳しく説明します。この章の内容は以下の通りです :

セクション 4.1: モーション・コントローラの種類

セクション 4.2: モーション制御モード

セクション 4.3: モータ・ドライバのインタフェース

セクション 4.4: 機械スイッチのインタフェース

セクション 4.5: カウンタ

セクション 4.6: コンパレータ

セクション 4.7: その他のモーション機能

セクション 4.8: 割り込み制御

セクション 4.9: マルチ・カード動作

4.1 モーション・コントローラの種類

サーボ / ステッパ・ドライバが最初にリリースされたとき、モータ制御はモータ制御とモーション制御の 2 つのレイヤーに分かれていました。モータ制御は PWM、パワー・ステージ、閉ループ、ホール・センサー、ベクトル空間などに関連し、モーション制御は速度プロファイルの生成、軌跡追従、多軸同期化、調整などを行います。

4.1.1 電圧タイプのモーション制御インタフェース

モーション制御とモータ制御間のインタフェースは急速に変化しています。電圧信号は当初からモータ・コントローラのコマンドとして使用されました。信号の振幅はモータの回転速度を表し、電圧の変化の間隔はある速度から別の速度へのモータの加速度を表します。モータ・ドライバのコマンドとしての電圧信号がいわゆる「アナログ」タイプのモーション・コントローラです。モータ・コントローラのアナログ回路への統合は極めて容易ですが、同タイプのモーション制御ではノイズが大きな問題となることがあります。また、モータの位置決め制御を行う場合、アナログ・タイプのモーション・コントローラは位置情報のフィードバック信号を必要とし、位置決め制御を可能にする閉ループ制御アルゴリズムを使わなければなりません。これにより、モーション制御はより複雑なものになりました。

4.1.2　パルス・タイプのモーション制御インタフェース

2 番目のモーションおよびコー他制御インタフェースのタイプはパルス・トレインです。デジタル・ワールドのトレンドとして、パルス・トレイン・タイプはモーション制御に新しいコンセプトを提供します。パルスのカウントはモータ回転のステップ数を表し、パルスの周波数はモータの動作速度を表します。また、周波数が変化する間隔はモータの加速度を表します。このインタフェースによって、ユーザーはアナログ・タイプの位置決めアプリケーションよりもはるかに容易にサーボまたはステッパ・モータを制御できます。これにより、モーション制御とモータ制御はより容易に分離できます。

上記の 2 つのインタフェースはゲイン調整を必要とします。アナログ位置コントローラでは、内部に制御ループが構築され、ユーザーはコントローラからゲインを調整できます。パルス・タイプの位置コントローラでは、モータ・ドライブの外部に制御ループが構築され、ユーザ—はドライバのゲインを調整する必要があります。

複数軸の動作では、モーション制御のほうがモータ制御よりも重要かもしれません。産業アプリケーションでは、信頼性が非常に重要な要素となります。モータ・ドライバ業者は高性能製品を製造し、モーション・コントローラ業者は強力で多機能なモーション・ソフトウェアを開発しています。ADLINK のマシンにそれら 2 つの製品を統合すれば、完璧なソリューションを実現できます。

4.1.3　ネットワーク・タイプのモーション制御インタフェース

ネットワーク・モーション・コントローラは最近開発されました。モータ・ドライバとモーション・コントローラ間のコマンドはアナログ信号でも、パルス信号でもなく、位置情報とモータ情報を含むネットワーク・パケットです。このタイプのコントローラはデジタル化およびパケット化されているので、高い信頼性を提供できます。モーション・コントローラはリアルタイムでなければならないので、ネットワークはサイクル時間が 1 ms 以下のリアルタイム機能を備えていなければなりません。Mitsubishi の SSCNET ネットワークはそうした速度条件に合致したネットワークの 1 つです。

4.1.4 ソフトウェア・リアルタイムのモーション制御カーネル

モーション制御カーネルに使用される方法には、DSP ベース、ASIC ベース、ソフトウェア・リアルタイム・ベースの 3 つの方法があります。

モーション制御システムには完全にリアルタイムな制御サイクルが必要で、コントローラの計算は同じサイクルで制御データを提供しなければなりません。そうでない場合、モータはスムーズに動作しません。通常、これは PC の計算性能およびフィードバック・カウンタ・カード、電圧出力またはパルス出力カードを使って行われます。この方法は非常にローエンドですが、広範なソフトウェア開発が求められます。システムでリアルタイム性能を確保するためには、リアルタイム・ソフトウェアを使用しなければなりません。システムの複雑性は増しますが、この方法はプロのモーション・コントローラ開発者にとっては最も柔軟な方法です。最も一般的な方法は NC マシンによる方法です。

4.1.5 DSP ベースのモーション制御カーネル

DSP ベースのモーション制御カーネルはコンピュータのリアルタイム・ソフトウェアの問題を解決します。DSP はマイクロ・プロセッサで、すべてのモーション制御計算は DSP で処理されます。DSP は専用の OS を使ってすべての手順を処理するので、リアルタイム・ソフトウェアの問題はありません。また、Windows ベースのコンピュータのような他の入力の割り込みやコンテキスト・スイッチの問題もありません。DSP はリアルタイム条件に対応した完全な性能を備えていますが、その計算速度は今のところ PC の CPU ほど速くはありません。また、DSP ベース・コントローラの業者とユーザーのインタフェースを結ぶソフトウェアは使用が簡単ではありません。ユーザーが学ぶアセンブリ言語を提供するコントローラ業者もあれば、ユーザーが使用するハンドシェーク・ドキュメントだけを提供するコントローラ業者もありますが、どちらも使用するのは容易ではありません。通常、DSP ベースのコントローラはマシン・メーカーがアプリケーションを構築するソフトウェア・カーネルよりも優れた方法を提供します。

4.1.6 ASIC ベースのモーション制御カーネル

ASIC ベースのモーション制御カーネルはソフトウェアや DSP カーネルとはかなり異なります。すべてのモーション機能は ASIC が行

うので、リアルタイムの問題はありません。ユーザーまたはコントローラ業者はASICが必要とする幾つかのパラメ―タを設定するだけで、簡単にモーション制御を実行できます。この種のモーション制御を使えば、システム統合のすべての問題は、モータ・ドライバの性能、ASIC が出力するプロファイル、ベンダーのソフトウェア・パラメータと ASIC の関係、ユーザーのコマンドとベンダーのソフトウェアの関係の 4 つの分野に分類されます。ASIC ベースのモーション制御はモーション・コントローラがデバイス間でよりスムーズに共同作業するのに役立ちます。

4.1.7 モーション制御の比較表

	ソフトウェア	ASIC	DSP
価格	* 普通	安価	高価
機能性	最高	低	通常
メンテナンス	煩雑	容易	普通

* リアルタイム OS を含む

	アナログ	パルス	ネットワーク
価格	高	低	** 通常
信号品質（距離参照）	普通	良	最良
メンテナンス	煩雑	普通	容易

** DSP またはソフトウェア・リアルタイム OS が必要

4.1.8 PCI-8158 のモーション・コントローラ・タイプ

PCI-8158 は ASIC ベース、パルス・タイプのモーション・コントローラです。このコントローラは、モーション ASIC、PCI カード、ソフトウェア・モーション・ライブラリの 3 つのブロックから構成されます。ユーザーは Windows 2000/XP、Linux、RTX ドライバ上で ADLINK のソフトウェア・モーション・ライブラリを使ってモーション ASIC にアクセスできます。ADLINK のソフトウェア・モーション・ライブラリはモータ制御のワンストップ機能を提供します。速度パラメータの計算はすべて ADLINK のライブラリを使って行われます。

例えば、台形速度プロファイルで 1 軸のポイント・ツー・ポイント・モーションを実行する場合、1 つの関数にターゲットの位置、

速度、加速時間を入力するだけです。その後、モータはプロファイル通りに動作します。各制御サイクルのパルスは ASIC が発生するので、CPU リソースは消費されません。モーション・コントローラは希望の速度プロファイルを使って正しいパルス・カウントを送信する責任を負っているだけなので、ターゲット位置の精度はモーション・コントローラのコマンドではなく、閉ループのコントロール性能およびモータ・ドライバの機械部分に依存します。したがって、プログラマー、機械エンジニア、電気エンジニアにとって、問題を検出して解決することは一層容易になります。

4.2 モーション制御モード

モータ制御は正または負の動きだけのものではありません。モーション制御を使えば、モータは特定の速度プロファイル、軌跡、他の軸との同期条件に従って動作できます。以下のセクションでは、このモーション・コントローラで実行可能なモーション制御モードについて説明します。

4.2.1 座標系

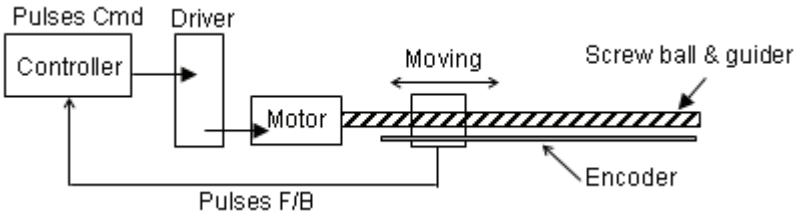
長さの単位にはデカルト座標系とパルスが使用されます。実際の長さは機械部品およびモータの解像度に依存します。例えば、モータがスクリュー・ボールに搭載されている場合、スクリュー・ボールのピッチは 10mm で、モータの回転に必要なパルスは 10,000 パルスになります。1 パルスの実際の単位は 10mm/10,000p =1 マイクロ・メータに等しいと言えます。

15,000 パルスのコマンドを設定すると、モーション・コントローラは 15mm 移動します。15.0001mm 移動させるにはどうすればよいでしょうか。モーション・コントローラは 1 パルス未満の残余値を保持して、それを次のコマンドに加えます。

モーション・コントローラは増加分のパルスをモータ・ドライバに送信します。これは、モータ・ドライバには相対コマンドだけが送信されるという意味です。ただし、この問題は現在の位置とターゲットの位置の違いを最初に計算することで解決できます。

その後、その違いをモータ・ドライバに送信します。例えば、現在の位置が 1000 で、モータを 9000 に移動させたい場合、絶対コマンドを使って、ターゲットの位置 9000 を設定できます。モーション・コントローラ内部では、最初に現在位置 1000 が取得されてから、ターゲットの位置との違いが計算されます。結果は +8000 となるので、モーション・コントローラはモータ・ドライバが 9000 の位置に移動するように 8000 パルスを送信します。

マシンの位置をチャックするために、直線スケールまたは外部エンコーダを取り付けなければならないときがあります。しかし、その座標系はどのように構築すればよいでしょうか。外部エンコーダの解像度は 10,000 パルス /1mm であるとします。モーション・コントローラが 1,000 パルスを送信すると、モータは 1mm 動きます。したがって、1 mm 動かす場合は、モータ・ドライバに 1,000 パルス送信する必要があります。その後、エンコーダのフィードバック値は 10,000 パルスになります。エンコーダの現在位置の読みが 3500 パルスで、絶対コマンドを使ってモータを 10,000 パルスの位置に移動させたい場合、モータ・ドライブに何パルス送信したらよいでしょうか。答えは (10000 - 3500) / (10,000 / 1,000) = 650 パルスです。「移動比」が設定されている場合、モーション・コントローラはそれを自動的に計算します。「移動比」はフィードバック解像度 / コマンド解像度に等しくなります。



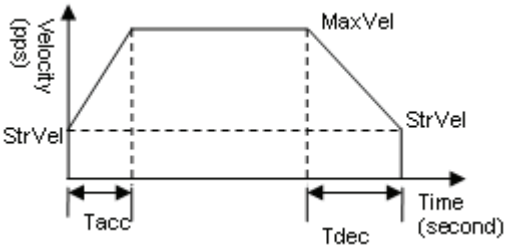
4.2.2 絶対的および相対的位置移動

座標系のターゲット位置を特定するには、絶対コマンドと相対コマンドの 2 種類のコマンドがあります。絶対コマンドはモーション・コントローラに位置を与え、モーション・コントローラはモータを現在の位置からその位置に移動させます。相対コマンドはモーション・コントローラを動かす距離を指定し、モーション・コントローラはモータを現在の位置からその距離移動させます。

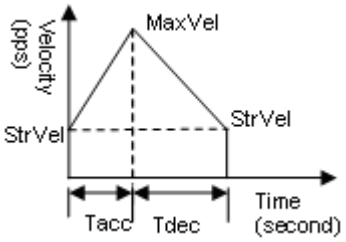
移動中の速度プロファイルを指定すれば、位置に到達するまでの速度を定義できます。

4.2.3 台形速度プロファイル

台形速度プロファイルでは、加速 / 減速エリアが 1 次直線の速度プロファイル（一定の加速率）に従います。プロファイル表は以下の通りです :



速度プロファイルの面積はこのモーションの距離を表します。希望の距離が与えられた速度パラメータの面積よりも小さい場合、プロファイルが三角形に見えるときがあります。その場合、モーション・コントローラは最大速度を下げますが、距離条件を満たす加速率を維持します。その場合の図は以下の通りです :



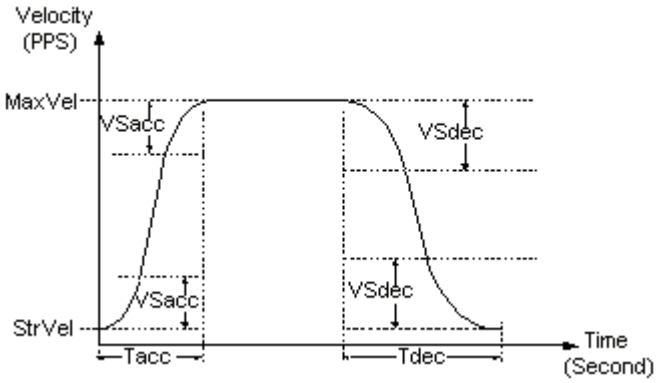
この種の速度プロファイルは速度モード、1 軸または複数軸の直線補間の位置モード、2 軸円形補間モードに適用されます。

4.2.4 S 曲線およびベル曲線速度プロファイル

S 曲線では、加速 / 減速エリアの速度プロファイルが 2 次曲線に従います。また、モータのスタートおよびストップ開始時に振動を低減できます。モーション中に加速 / 減速を拡大するには、それらのエリアに直線部分を挿入する必要があります。この形は「ベ

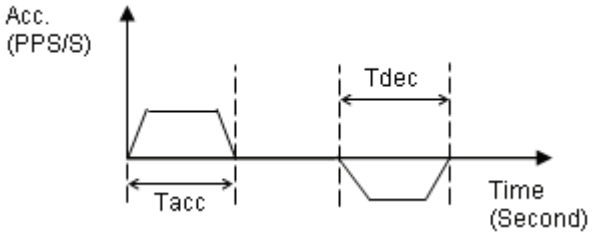
ル」曲線と呼ばれます。S 曲線の上部と下部の間に直線が追加されます。この形は加速の速度を増すと共に、加速の振動を低減します。

ベル曲線のパラメータは以下のように定義されます :



- Tacc: 加速時間（秒単位）
- Tdec: 減速時間（秒単位）
- StrVel: 初速度（PPS 単位）
- MaxVel: 最大速度（PPS 単位）
- VSacc: 加速時のベル曲線の S 曲線部分（PPS）
- VSdec: 減速時のベル曲線の S 曲線部分（PPS）

VSacc=0 または VSdec=0 の場合、加速または減速は直線部分のない純粋な S 曲線となります。ベル曲線の加速図は以下の通りです :

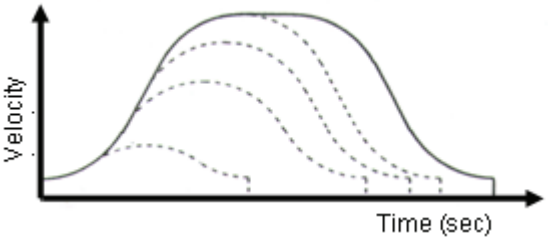


S 曲線プロファイルのモーション関数は常にスムーズなモーションを発行するように設計されています。最終位置とリンクした加速度パラメータの時間が原因で、軸が最大速度に到達できない（す

なわち、MaxVel に達するには距離が短すぎる）場合、最大速度は自動的に低下します。（下図参照）

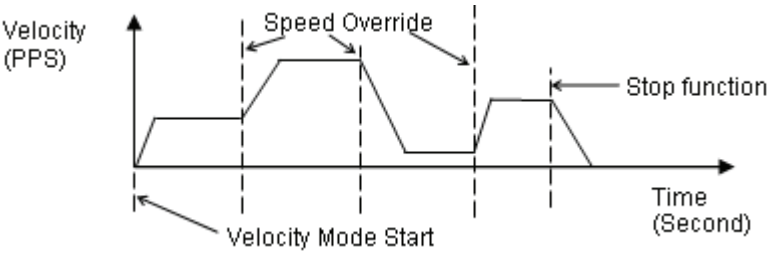
通常、MaxVel および Tacc、Tdec、VSacc、VSdec の値は自動的に低下し、StrVel、加速度、加加速度は変わりません。これは台形プロファイル・モーションにも適用されます。

この種の速度プロファイルは速度モード、1 軸または複数軸の直線補間の位置モード、2 軸円形補間モードに適用されます。



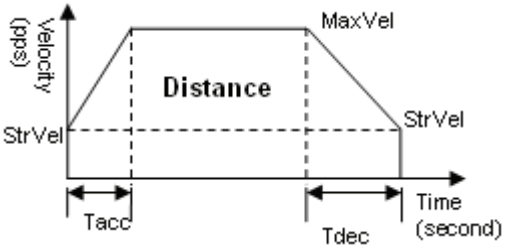
4.2.5 速度モード

加速モードでは、停止コマンドが発行されるまで、パルス・コマンドが連続して出力されます。他の理由で停止されない限り、モータはターゲット位置や希望の距離に関係なく動作します。出力パルスは初速度から指定の最大速度に達するまで加速し、加速直線または S 曲線に従います。また、他の速度コマンドが設定されたり、停止コマンドが発行されたりするまで、パルス出力レートは最大速度を維持します。速度は新しい速度設定でオーバーライドできます。新しい速度に最初の動作速度と反対の速度は指定できません。この種のモーションの速度プロファイルは以下の通りです：



4.2.6　1 軸位置モード

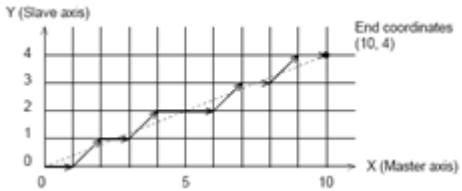
位置モードでは、モーション・コントローラは希望の位置または距離に等しい量のパルスを出力します。距離または位置の単位はモーション・コントローラ内のパルスです。距離の最小長は 1 パルスです。PCI-8158 では、ユーザーがパルスの実際の長さを変換できる浮動点関数を使用できます。ADLINK のソフトウェア・ライブラリでは、1 パルス未満の距離をレジスタ内に保持し、次のモーション関数に適用します。ADLINK のモーション・コントローラはパルス・カウントによる位置決め以外にも、1 次台形、2 次 S 曲線、それらを混在したベル曲線の 3 種類の速度プロファイルを提供して、位置決めを行います。ユーザーはそれぞれの関数を呼び出して、位置決めを実行できます。距離と速度プロファイルの関係は下図の通りです :



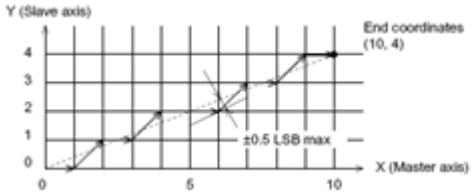
このプロファイルの V-t 図の面積が距離となります。

4.2.7　2 軸直線補間位置モード

「複数軸間の補間」では、複数軸が同時に出発し、最終地点に同時に到達します。直線では、各軸の速度比が一定です。(0,0) から (10,4) へのモーションを実行すると、直線補間の結果は下図のようになります。



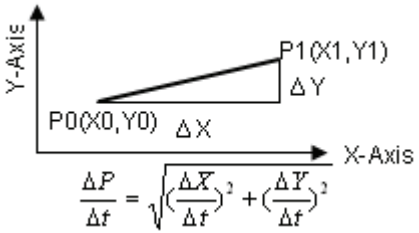
X 軸または Y 軸からのパルス出力は完全な直線に従って 1/2 パルスの違いを残します。直線補間の精度は以下の通りです :



補間グループを停止するには、グループの最初の軸に停止関数を呼び出します。

下図のように、8 軸直線補間では、XY の位置が P0 から P1 に移動します。2 つの軸は同時に出発および停止し、軌跡は直線になります。

X 軸と Y 軸の速度比は (ΔX: ΔY) で、ベクトル速度はこうなります :

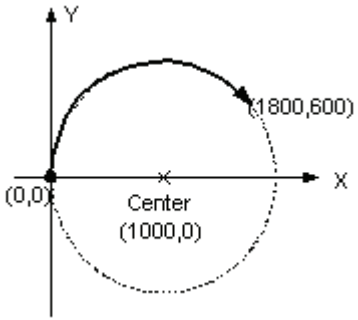


8 軸直線補間関数を呼び出す場合、ベクトル速度は初速度 StrVel と最大速度 MaxVel を定義する必要があります。

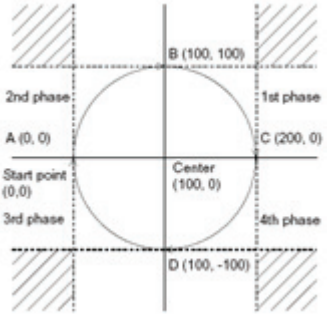
4.2.8 2 軸円形補間モード

円形補間では、XY 軸は出発点 (0,0) から同時に出発し、終点 (1800,600) で停止します。両点を結ぶ軌道は円弧で、MaxVel は接

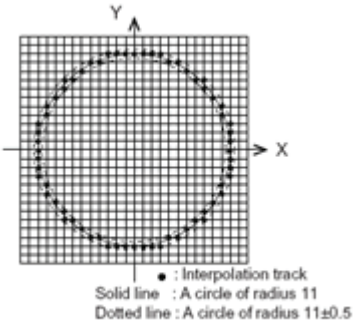
線速度です。円弧の終点が正しい位置ではない場合は停止せずに円形に移動します。



終点が円弧の軌道上にない場合でも、モーション・コントローラはユーザーが希望する終点に移動します。ただし、終点が以下のグラフの斜線部分にないと、停止せずに円形に移動し続けます。



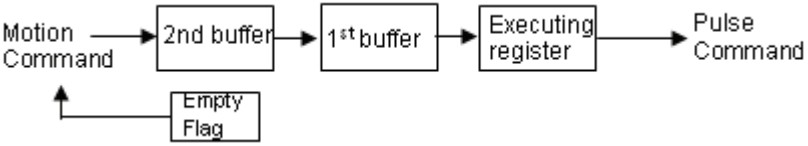
円形補間のコマンド精度は以下の通りです。誤差は半径± 1/2 パルスです。



4.2.9 連続モーション

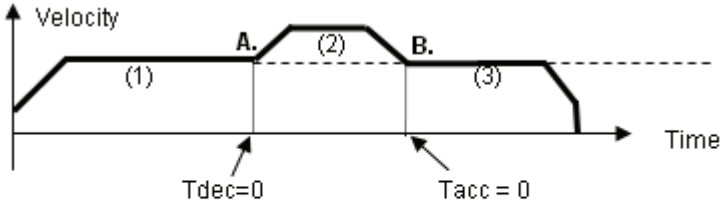
連続モーションでは、一連のモーション・コマンドまたは位置を連続して実行できます。中断することなく、前のコマンドの直後に新しいコマンドを設定できます。モーション・コントローラには 3 つのコマンド・バッファ（プリレジスタ）が内蔵されているので、連続モーションが可能となります。

最初のコマンドを実行していても、2 番目のコマンドを最初のバッファに、3 番目のコマンドを 2 番目のバッファに設定できます。最初のコマンドが完了すると、モーション・コントローラは 2 番目のコマンドを実行レジスタに、3 番目のコマンドを最初のバッファに移します。すると、2 番目のバッファが空くので、4 番目のコマンドを 2 番目のバッファに設定できます。通常、実行レジスタが完了する前に新しいコマンドを 2 番目のバッファに設定する時間がある場合、モーションをエンドレスで実行できます。下図は連続モーションのこの構造を示しています。



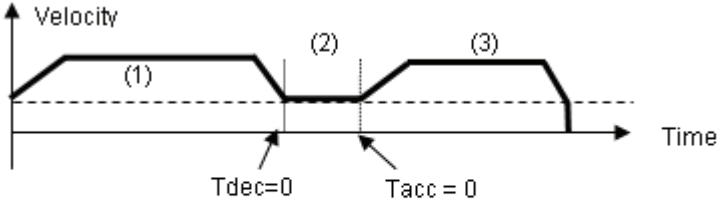
速度連続プロファイルを実行するには、位置コマンドに加え、速度コマンドを正しく設定しなければなりません。以下の例では、

この連続モーションの 3 つのモーション・コマンドがあります。2 番目のコマンドは他よりも高速です。これら 3 つのモーション関数の間の速度の相互接続は下図のように設定する必要があります :



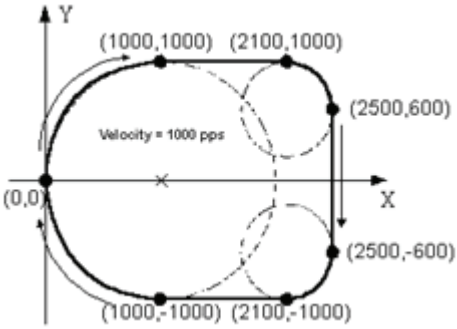
Tdecof the first command =0
StrVel of the second command = MaxVel of the first command
Taccof the third command =0
MaxVel of the third command = StrVel of the second command

2 番目のコマンドの速度値が他を下回る場合、設定は下図のようになります :



Taccc of the second command =0
Tdec of the second command =0
MaxVel of the second command = StrVel of the first command
MaxVel of the second command = StrVel of the second command

8 軸連続円弧補間でも状況は変わりません。2 つのコマンドの速度設定が合致するように速度を設定できます。



INP チェックを有効にすると、モーションはバッファ内の各コマンド間に遅延を生じさせます。INP チェックを有効にすると、希望の地点に到達できますが、各コマンド間の円滑性は減少します。遅延を必要とせず、スムーズなモーションが必要な場合は INP チェックを無効にしてください。

4.2.10 原点復帰モード

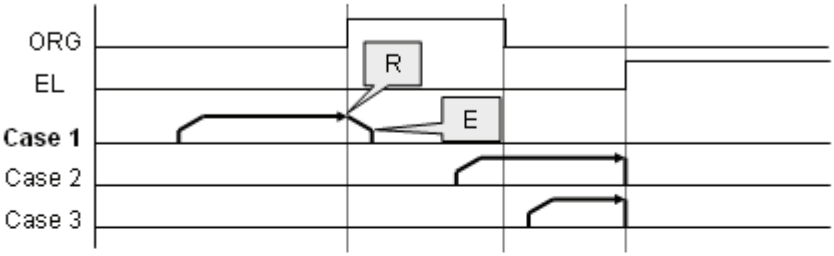
原点復帰とは、座標のゼロ位置点を検索することです。座標のゼロ位置として ORG、EZ、または EL ピンを使用する場合もあります。システム起動時、プログラムはこのマシンのゼロ点を検出する必要があります。ADLINK のモーション・コントローラはゼロ点を検出する原点復帰モードをサポートしています。

ADLINK では多数の原点モードをサポートしており、各モードには多数の制御相が含まれます。それらの相は全て ASIC によって実行されます。ソフトウェアを追加する必要もなければ、CPU に負荷がかかることもありません。原点復帰が完了すると、ターゲットのカウンタは ORG 入力時のライジング・エッジといった希望の原点モードでゼロにリセットされます。モーション・コントローラはカウンタ・リセット後、マシンを減速させるためにパルスを出力し続けることがあります。モータが停止すると、カウンタはゼロ点ではないかもしれませんが、原点復帰手順は完了しています。表示されるカウンタ値は参照位置で、マシンはすでに原点復帰しています。

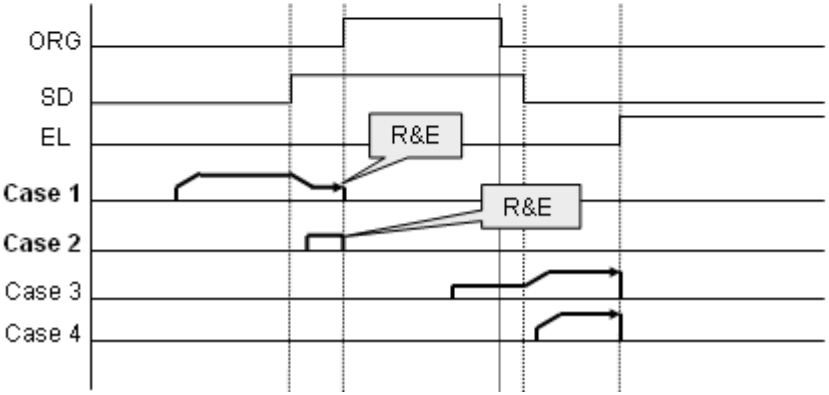
以下の図で様々な原点モードを紹介します。R はカウンタ・リセット（コマンドおよび位置カウンタ）、E は ERC 信号出力をそれぞれ意味します。

Home mode=0: (ORG Turn ON then reset counter)

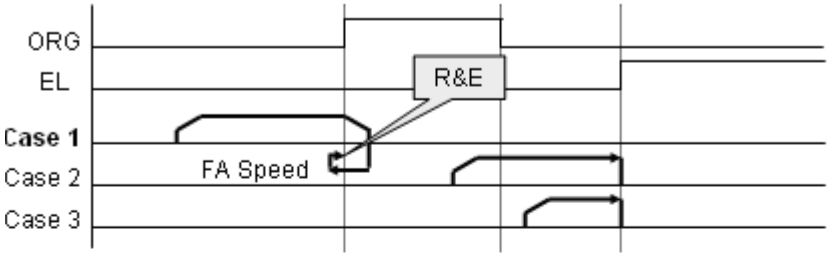
- When SD is not installed



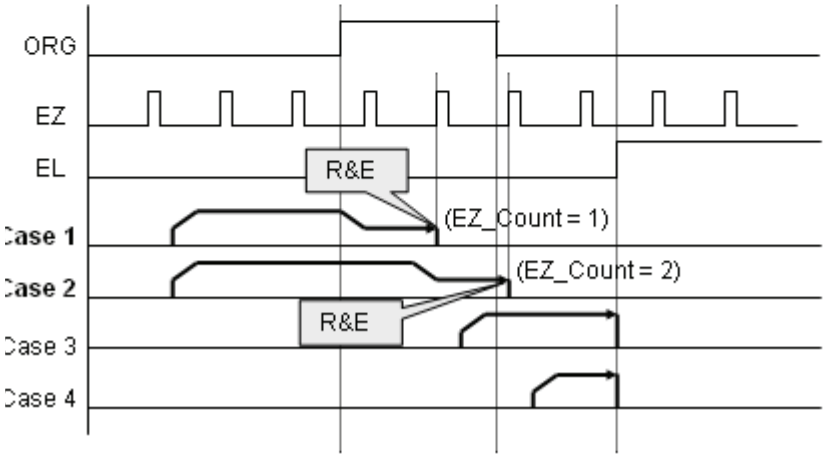
- When SD is installed and SD is not latched



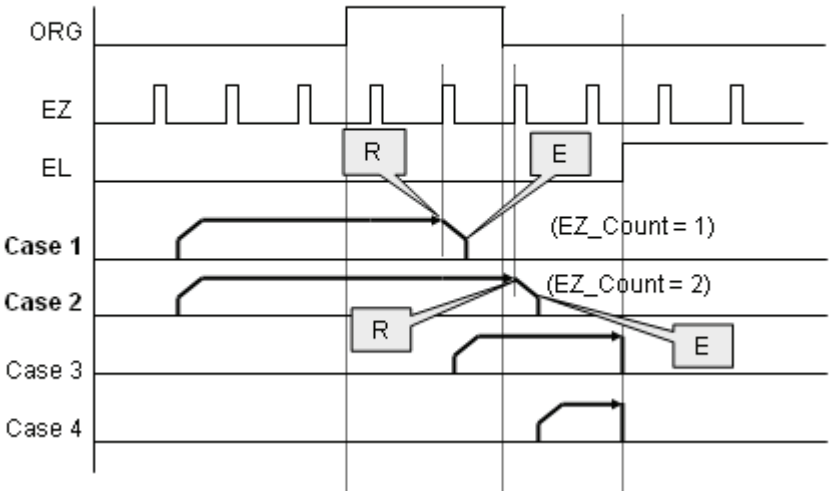
Home mode=1: (Twice ORG turn ON then reset counter)



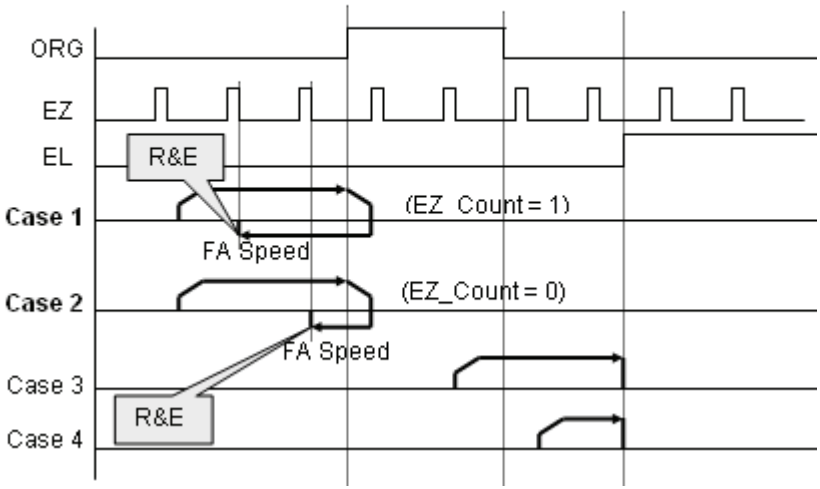
Home mode=2: (ORG ON then Slow down to count EZ numbers and reset counter)



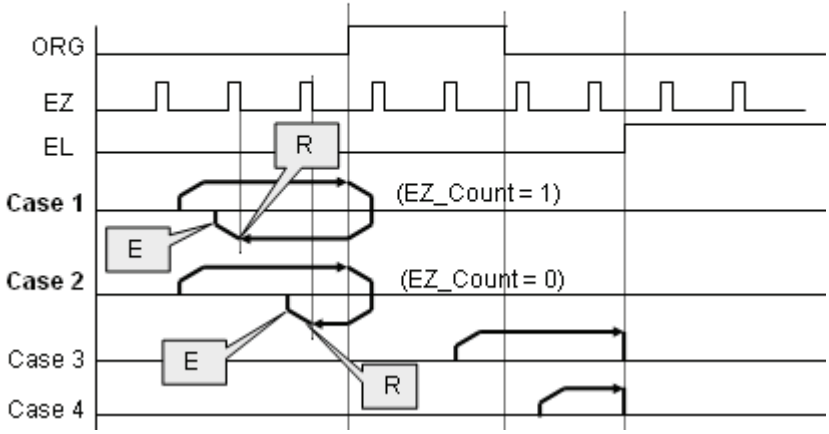
Home mode=3: (ORG On then count EZ numbers and reset counter)



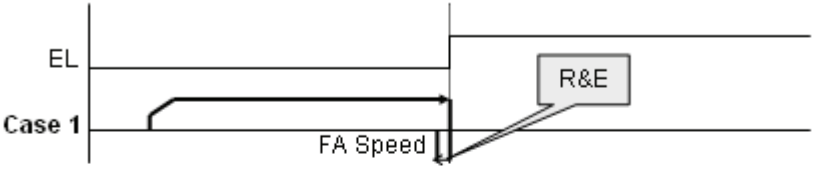
Home mode=4: (ORG On then reverse to count EZ number and reset counter)



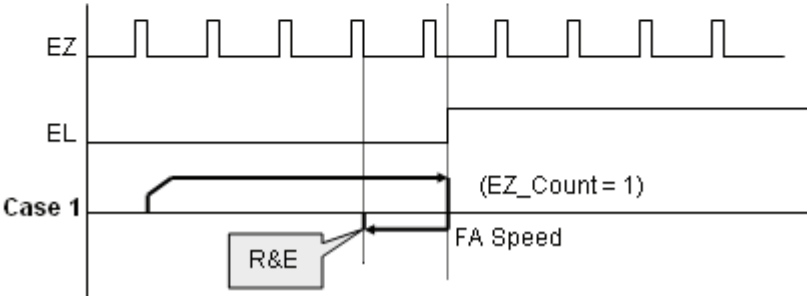
Home mode=5: (ORG On then reverse to count EZ number and reset counter, not using FA Speed)



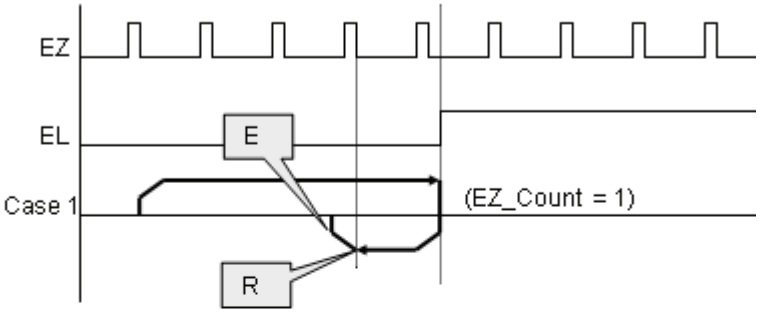
Home mode=6: (EL On then reverse to leave EL and reset counter)



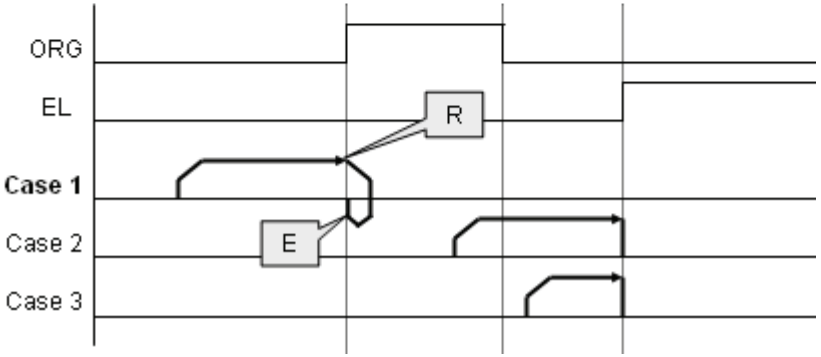
Home mode=7: (EL On then reverse to count EZ number and reset counter)



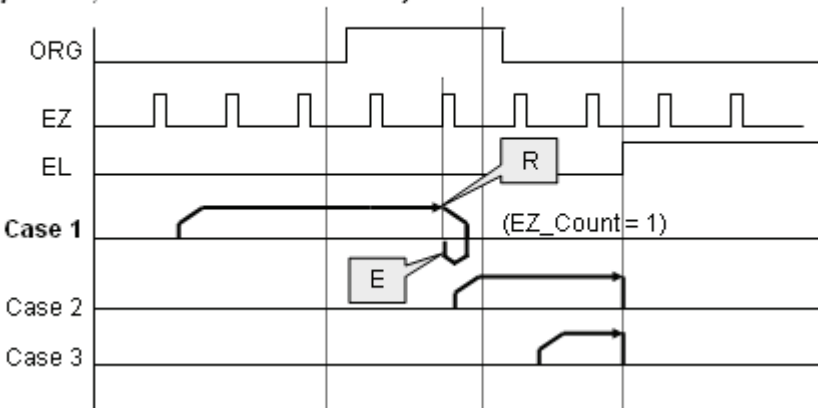
Home mode=8: (EL On then reverse to count EZ number and reset counter, not using FA Speed)



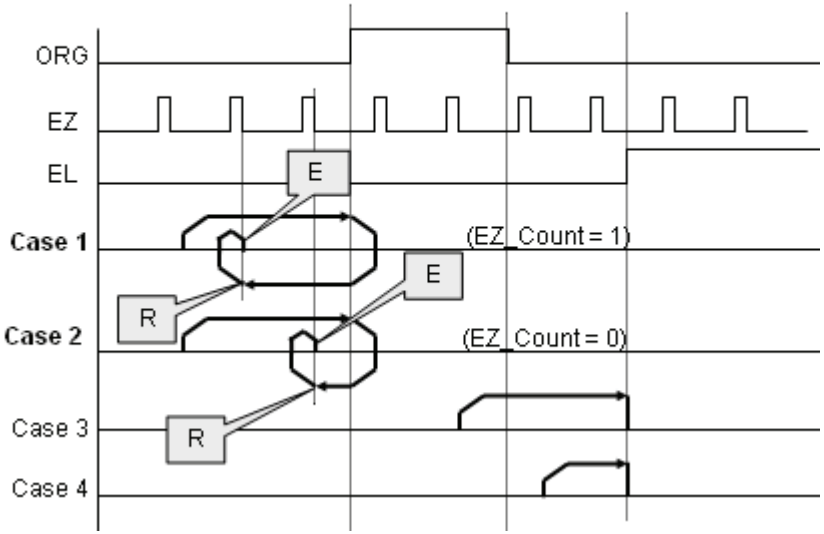
Home mode=9: (ORG On then reverse to zero position, an extension from mode 0)



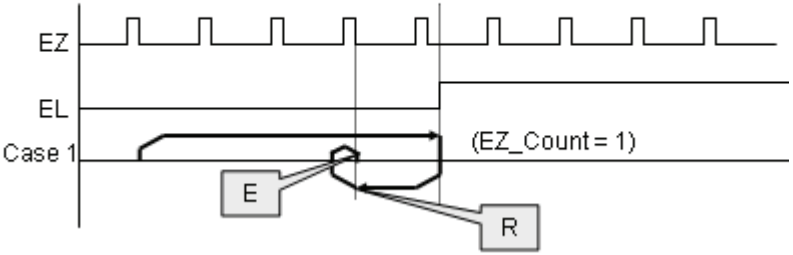
Home mode=10: (ORG On then counter EZ and reverse to zero position, an extension from mode 3)



Home mode=11: (ORG On then reverse to counter EZ and reverse to zero position, an extension from mode 5)

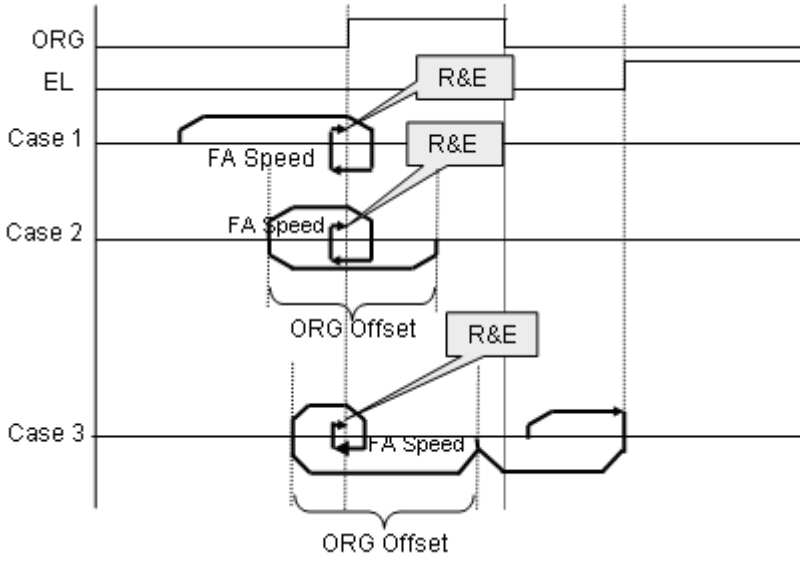


Home mode=12: (EL On then reverse to count EZ number and reverse to zero position, an extension from mode 8)



4.2.11 原点検索機能

このモードは軸の位置に関わらず、前セクションで説明されている通常の原点復帰モードに自動検索機能を追加するのに使用されます。下図は原点検索機能を使った原点モード 2 の例です。ORG オフセットはゼロになり得ません。提案値は ORG の面積の 2 倍の長さです。

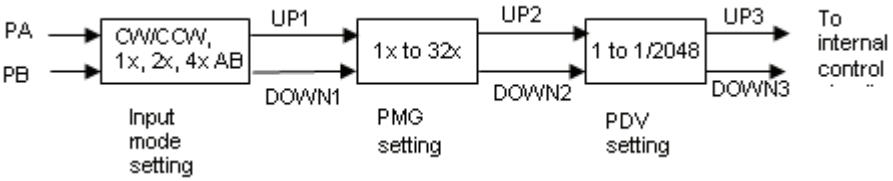


4.2.12 手動パルス機能

手動パルスは手動でパルス・トレインを発生させる装置です。パルスはモーション・コントローラに送信されてから、パルス出力ピンに送られます。入力パルスは送出前に増幅したり、分割したりできます。

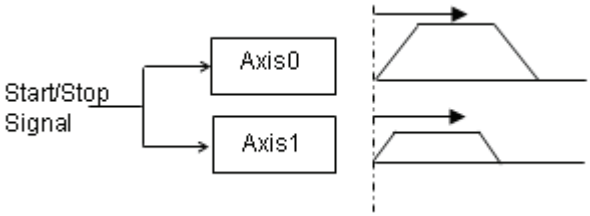
モーション・コントローラは手動パルス装置から CW/CCW と AB 相の 2 種類のパルス・トレインを受信します。AB 相の入力モードを選択すると、1 倍、2 倍、または 4 倍の選択肢が追加されます。

パルス比のブロック図は以下の通りです。



4.2.13 同時スタート機能

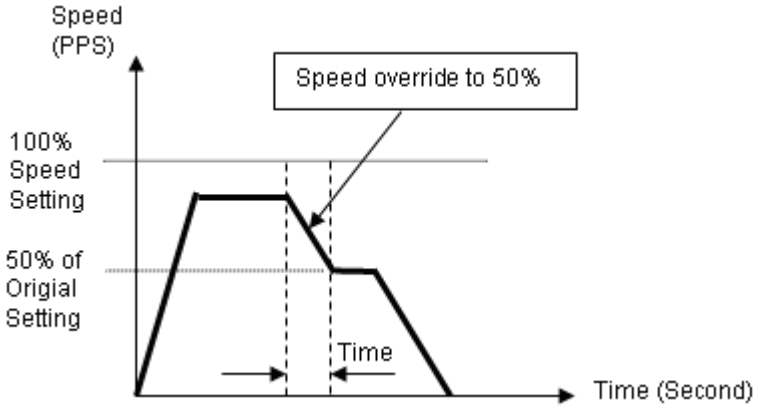
同時モーションでは、複数の軸が外部信号でも、内部信号でもよい同時信号でスタートできます。外部信号では、すべての軸の移動パラメータを最初に設定しなければなりません。その後、それらの軸は外部スタート / ストップ・コマンドを待って、出発または停止します。内部信号では、スタート・コマンドはソフトウェアのスタート機能から送信されます。コマンドが発行されると、同期モードで待機しているすべての軸が同時にスタートします。



4.2.14 速度オーバーライド機能

速度オーバーライドでは、モーション実行時にコマンドの速度を変更できます。変更パラメータは最初に定義した速度の百分率で

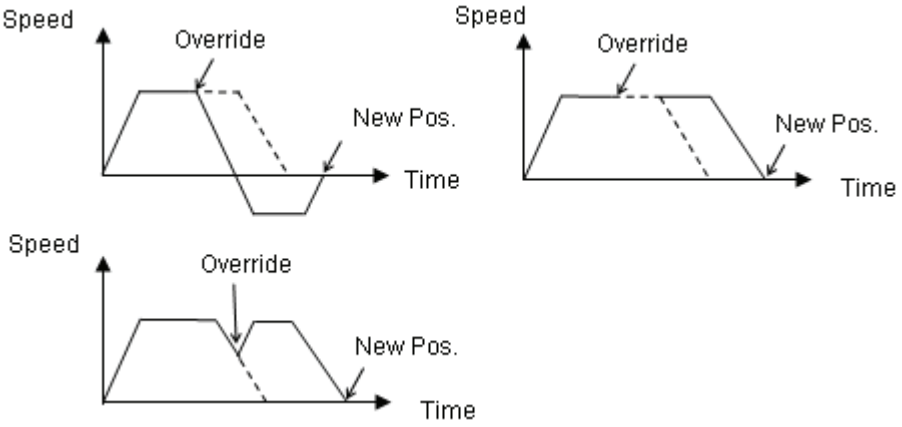
表されます。100% の速度値を定義してから、モーション実行時に最初の速度の百分率で速度を変更できます。ユーザーが 100% の速度値を定義していない場合、デフォルトの 100% の速度は最後のモーション・コマンドの最大速度になります。この機能はすべてのモーション機能に適用されます。実行モーションが S 曲線またはベル曲線の場合、速度オーバーライドは純粋な S 曲線になります。実行モーションが T 曲線の場合、速度オーバーライドは T 曲線になります。



4.2.15 位置オーバーライド機能

位置オーバーライドでは、実行中に位置決めコマンドを発行して、ターゲットの位置を変更できます。オーバーライド・コマンド発行時、新しいターゲット位置が現在の位置に隠れている場合、モータは減速して、新しいターゲット位置に反転します。新しいターゲット位置が現在の位置から同じ方向に離れている場合、モーションは速度を維持して、新しいターゲット位置に向かいます。オーバーライドのタイミングが現在のモーションの減速時で、ターゲット位置が現在の位置から同じ方向に離れている場合、モーションは最初の速度に加速して、新しいターゲット位置に向かいます。以下の図は動作の例です。新しいターゲット位置の相対

パルスが最初の減速パルスを下回っている場合、この機能は正しく動作できないことに注意してください。



4.3 モータ・ドライブのインタフェース

ADLINK はモータ・ドライブに直接接続可能で、独自の機能をもつ複数の専用 I/O を提供しています。モータ・ドライバは外部モーション・コントローラが使用可能な多数の I/O ピンを備えています。それらの I/O ピンは、パルス・コマンドやエンコーダ・インタフェースなどのパルス I/O 信号とサーボ・オン、アラーム、INP、サーボ・レディ、アラーム・リセット、緊急停止入力などのデジタル I./O 信号の 2 つのグループに分類されます。以下のセクションではそれらの I/O ピンの機能について説明します。

4.3.1 パルス・コマンド出力インタフェース

モーション・コントローラはパルス・コマンドを使って、モータ・ドライバからサーボ / ステッパ・モータを制御します。ドライバはパルス・トレインを位置コマンドとして受信できる位置モードに設定します。パルス・コマンドは 2 つの信号ペアから構成され、コネクタの OUT および DIR ピンとして定義されます。各信号は差動出力のペアとして 2 つのピンを有しています。パルス出力コマンドには、(1) シングル・パルス出力モード (OUT/DIR) と (2) デュアル・パルス出力モード (CW/CCW タイプ・パルス出力) の 2 つの信号モードがあります。モードはモータ・ドライバのモー

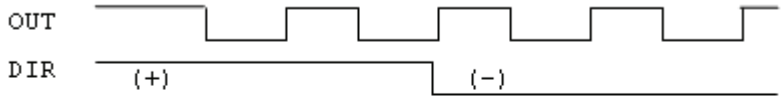
ドと同じでなければなりません。モードと OUT および DIR ピンの信号タイプの関係は下表の通りです：

シングル・パルス出力モード（OUT/DIR モード）

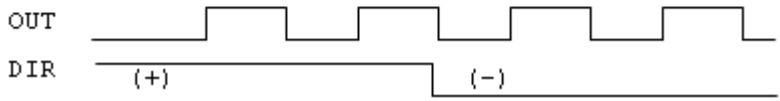
モード	OUT ピンの出力	DIR ピンの出力
デュアル・パルス出力（CW/CCW）	プラス（すなわち CW）方向のパルス信号	マイナス（すなわち CCW）方向のパルス信号
シングル・パルス出力（OUT/DIR）	パルス信号	方向信号（レベル）

このモードでは、OUT ピンはコマンド・パルス・チェーンの出力に使用されます。OUT パルスの番号はパルスの距離を表します。OUT パルスの周波数はパルスの秒速を表します。DIR 信号は正（+）または負（−）のコマンド方向を表します。出力波形は下図の通りです。パルス・チェーンの極性は設定可能です。

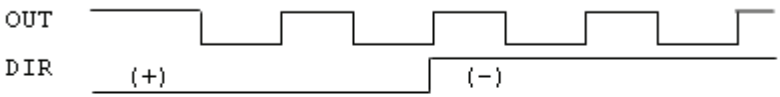
Pulse mode = 0: (OUT pin normally high)



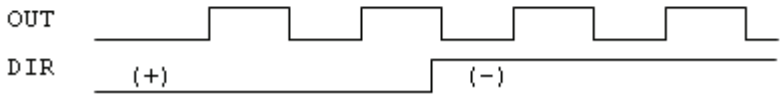
Pulse mode = 1: (OUT pin normally low)



Pulse mode = 2: (OUT pin normally high)



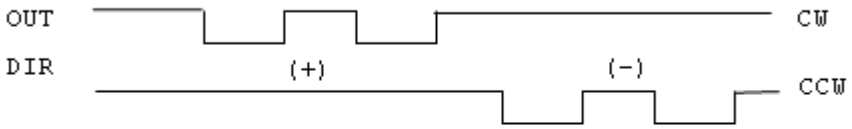
Pulse mode = 3: (OUT pin normally low)



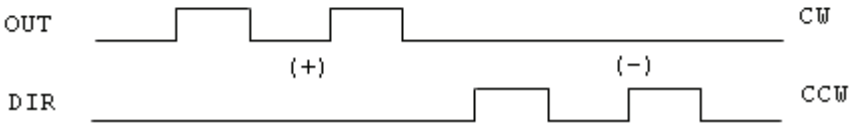
デュアル・パルス出力モード（CW/CCW モード）

このモードでは、OUT および DIR ピンの波形は CW（時計回り）および CCW（反時計回り）のパルス出力をそれぞれ表します。パルスの番号はパルスの距離を表します。パルスの周波数はパルスの秒速を表します。CW ピンのパルス出力はモータを正の方向に動かし、CCW からのパルス出力はモータを負の方向に動かします。正（+）コマンドと負（−）コマンドの出力波形は下図の通りです。

Pulse outmode = 4: (Pulse is normally high)



Pulse outmode = 5: (Pulse is normally low)



コマンドのパルスは 28-bit のコマンド・カウンタでカウントされます。コマンドのカウンタはコントローラからのパルス出力の合計値を保存できます。

4.3.2 パルス・フィードバック入力のインタフェース

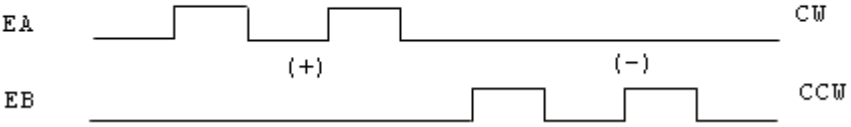
ADLINK のモーション・コントローラは各軸に 1 つの 28-bit アップ / ダウン・カウンタを備えて、パルスのフィードバックをカウントします。これは位置カウンタと呼ばれます。位置カウンタはコネクタに差動信号入力用プラスおよびマイナス・ピンを備えた EA および EB 信号からのパルスをカウントします。また、デュアル・パルス入力（CW/CCW モード）と AB 相入力の 2 つのパルス・タイプを受信します。AB 相入力は 1 倍、2 倍、または 4 倍に増幅できます。インクリメンタル・エンコード入力で最も一般的に使用されるのは 4 倍 AB 相モードです。

例えば、回転エンコーダの 1 回転が 2000 パルスである場合、AB 相を 4 倍すると、位置カウンタのカウンタ値の表示は 1 回転あたり 8000 パルスになります。

モーション・コントローラにエンコーダを使用しない場合、このカウンタのフィードバック・ソースをパルス・コマンド出力に設定してください。設定しないと、カウンタ値は常にゼロになります。カウンタのフィードバック・ソースをパルス・コマンド出力に設定すると、コマンド出力パルスからのフィードバック・パルスが内部でカウントされるので、パルス・コマンド出力カウンタの位置カウンタ値を取得できます。

上の 2 種類のパルス・フィードバック信号は下図ようになります。

Plus and Minus Pulses Input Mode (CW/CCW Mode)



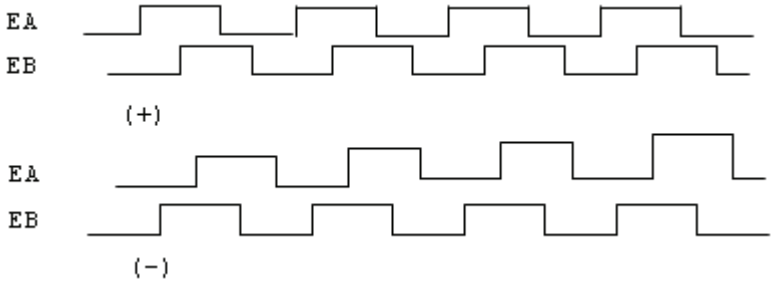
このモードのパルスのパターンは、入力ピンが EA および EB であることを除き、パルス・コマンド出力のセクションのデュアル・パルス出力モードの場合と同じです。

このモードでは、カウンタは EA ピンからのパルスでカウントアップし、EB ピンからのパルスでカウントダウンします。

90° 位相が異なる信号入力モード（AB 相モード）

このモードでは、EA 信号の位相は EB 信号と比較して 90° 進んでいるか、遅れています。2 つの信号の「進んでいる」または「遅れている」という位相の違いはモータの転換方向に起因します。EA 信号の位相が EB 信号の位相よりも進んでいる場合、アップ / ダウン・カウンタはカウントアップします。

波形は下図のようになります。

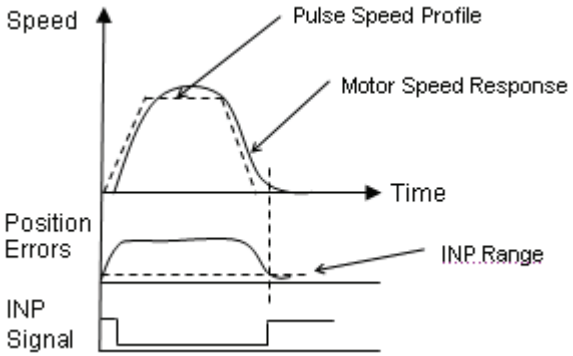


インデックス入力（EZ）信号は直線または回転エンコーダのゼロ参照として使用されます。EZ はマシンの機械的ゼロ位置を定義するのに使用できます。正しい結果を得るため、信号のロジックも正確に設定してください。

4.3.3 インポジション信号

インポジション信号はモータ・ドライバの出力信号で、モータが事前に定義したエラーを伴って位置に到達したことをモーション・コントローラに知らせます。事前に定義したエラーの値がインポジション値です。ほとんどのモータ・ドライバはそれを INP 値と呼んでいます。モーション・コントローラが位置決めコマンドを発行すると、INP 信号がオンになるまで、モーションのビジー・ステータスは「真」を維持します。モーションのビジー・フラッグとなる INP チェックは無効にできます。INP チェックを無効に

すると、すべてのパルス・コマンドの送信が終わるまで、モーションのビジーは「偽」となります。



4.3.4 サーボ・アラーム信号

アラーム信号はモータ・ドライバの出力信号で、サーボ・モータやドライバ内部でエラーが発生したことをモーション・コントローラに知らせます。モーション・コントローラがこの信号を受信すると、パルス・コマンドは送信を停止し、ALM 信号のステータスはオンになります。アラームの原因では、サーボ・モータのオーバー・スピード、過電流、過負荷などが考えられます。詳しくはモータ・ドライバのマニュアルを確認してください。

アラーム信号のロジックは正しく設定してください。アラーム・ロジックの設定とモータ・ドライバの設定が異なる場合、ALM ステータスは常にオンになって、パルス・コマンドは出力されません。

4.3.5 エラー・クリア信号

ERC 信号はモーション・コントローラの出力で、エラー・カウンタをクリアするようにモータ・ドライバに指示します。エラー・カウンタはコマンド・パルスとフィードバック・パルスの違いでカウントされます。フィードバック位置はコマンド位置よりも必ず遅れるので、それらの 2 つの位置の間には常にパルスの違いがあります。違いはエラー・カウンタに表示されます。モータ・ドライバはエラー・カウンタを基本的な制御インデックスに使用します。エラー・カウンタ値が大きくなるにつれ、モータ速度コマ

ンドは高速に設定されます。エラー・カウンタがゼロの場合、ゼロのモータ速度コマンドが設定されます。

以下の 4 つの状況では、エラー・カウンタを同時にクリアするため、ERC 信号がモーション・コントローラからモータ・ドライバに自動的に出力されます。

1. 原点復帰完了時
2. エンド・リミット・スイッチ接触時
3. アラーム信号アクティブ時
4. 緊急停止コマンド発行時

4.3.6 サーボ・オン / オフ・スイッチ

サーボ・オン / オフ・スイッチはモーション・コントローラの汎用デジタル出力信号で、コネクタ上の SVON ピンとして定義されます。また、モータ・ドライバの制御状態を切り替えるのに使用できます。同スイッチをオンにすると、ドライバの制御ループがアクティブになるので、モータはホールドされます。軸が垂直にインストールされており、サーボ信号がオフになっている場合、軸が未制御状態にになっていると、落下することがあるので注意してください。サーボ・アラームや緊急信号がオンの場合といった一部の状況でも同様です。

4.3.7 サーボ・レディ信号

サーボ・レディ信号はモーション・コントローラの汎用デジタル入力で、モーション・コントローラに関連した目的はありません。この信号とモータ・ドライバの RDY 信号を接続すれが、モータ・ドライバがレディ状態にあるかどうか確認できます。また、モータ・ドライバの電源が入っているかどうかもチェックできます。このピンは汎用入力として他の用途で接続できますが、モーション制御に影響を与えることはありません。

4.3.8 サーボ・アラーム・リセット・スイッチ

サーボ・ドライバ内部に問題があると、サーボ・ドライバはアラーム信号を発行します。アラームの原因には、サーボ・モータの過電流、オーバー・スピード、過負荷などがあります。電源のリセットでアラーム・ステータスをクリアできますが、通常、実行中はサーボ・モータの電源を切りたいとは思いません。サーボ・ドラ

イバにはユーザーがアラーム・ステータスをリセットできるピンが 1 つ用意されています。ADLINK のモーション・コントローラは各軸に 1 つの汎用出力ピンを備えています。このピンはサーボ・アラーム・ステータスのリセットに使用できます。

4.4 機械スイッチのインタフェース

ADLINK では、原点スイッチ（ORG）、プラスおよびマイナスのエンド・リミット・スイッチ（□}EL）、減速スイッチ（SD）、位置決め開始スイッチ（PCS）、カウンタ・ラッチ・スイッチ（(LTC）、緊急停止入力（EMG）、カウンタ・クリア・スイッチ（CLR）といった機会スイッチ専用の入力ピンを提供しています。これらのスイッチの応答時間は非常に短く、数 ASIC クロック・サイクルに過ぎません。上記信号の使用時、リアルタイムの問題はありません。すべての機能はモーション ASIC によって実行されます。ソフトウェアは何もする必要がなく、必要なのは結果を待つだけです。

4.4.1 原点信号

ADLINK のコントローラは各軸に 1 つの原点信号を備えています。この信号は軸のゼロ位置を定義するのに使用されます。原点手順を実行する前に、信号のロジックを正しく設定しなければなりません。詳しくは原点モードのセクションを参照してください。

4.4.2 エンド・リミット・スイッチ信号

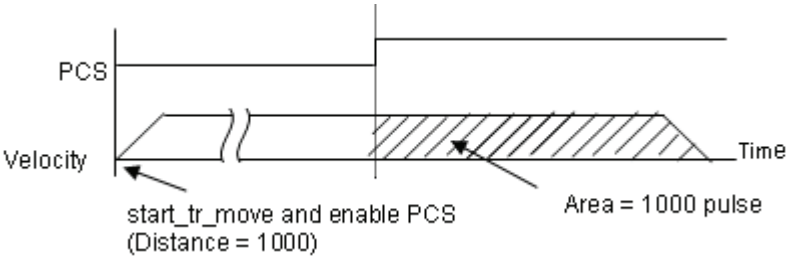
通常、エンド・リミット・スイッチは軸の両端に設置されます。軸の正位置にはプラス EL を、負位置にはマイナス EL を設置しなければなりません。これら 2 つの信号は安全上の理由から設置されます。反対に設置すると、保護が無効になります。モータの稼働部分がエンド・リミット信号の 1 つに接触すると、モーション・コントローラはパルスの送信を停止して、ERC 信号を出力します。これは操作を間違えたときにマシンが損傷するのを防ぎます。

4.4.3 減速スイッチ

減速信号はコマンド・パルスに初速度を強制的に減速させるのに使用されます。同信号は高速移動時の機械的稼働部品をメカニズムの限界から守るのに使用されます。SD 信号は正負の両方の方向で有効です。

4.4.4 位置決め開始スイッチ

位置決め開始スイッチは特定の位置を移動させるのに使用されます。機能は下図の通りです。



4.4.5 カウンタ・クリア・スイッチ

カウンタ・クリア・スイッチはモーション・コントローラのカウンタをリセットする入力信号です。外部コマンドによってカウンタをリセットする必要がある場合は、このピンを制御ソースに使用します。

4.4.6 カウンタ・ラッチ・スイッチ

カウンタ・ラッチ・スイッチはカウンタ値をレジスタに保存させる入力信号です。ある入力の投入時のカウンタ値を知る必要がある場合は、このピンを接続すれば取得できます。

4.4.7 緊急停止入力

ADLINK のモーション･コントローラは緊急時に備えた包括的なデジタル入力をサポートしています。入力をオンにすると、ADLINK のモーション・コントローラはマシンの損傷を防ぐため軸のすべてのモーションを直ちに停止します。通常、緊急停止ボタンはマシンのこの入力に接続します。この入力は安全上常時閉路式として使用するようお勧めします。

4.5 カウンタ

このモーション・コントローラの各軸には 4 つのカウンタがあります。このセクションではそれらのカウンタについては説明します。

- コマンド位置カウンタ : 出力パルスの数をカウントします。
- フィードバック位置カウンタ : 入力パルスの数をカウントします。
- 位置エラー・カウンタ : コマンドおよびフィードバック・パルス数の間のエラーをカウントします。
- 汎用カウンタ : ソースはコマンド位置、フィードバック位置、手動パルス、または ASIC クロックの半分に設定できます。
- ターゲット位置レコーダ : 最後のモーション・コマンドのソフトウェアに記録されたターゲット位置の値

4.5.1 コマンド位置カウンタ

コマンド位置カウンタは 28-bit バイナリのアップ / ダウン・カウンタです。その入力ソースはモーション・コントローラからの出力パルスです。コマンド位置カウンタは現在のコマンド位置の情報を提供し、モーション・システムのデバッグに役立ちます。

ADLINK のモーション・システムは開ループ式です。モータ・ドライバはモーション・コントローラからパルスを受信して、モータを駆動します。ドライバは非稼働時、このコマンド・カウンタをチェックして、更新値がないかどうか確認します。更新値がある場合、パルスが送信されていることを意味し、モータ・ドライバに問題が発生している場合があります。その場合は、モータ・ドライバのパルス受信カウンタをチェックしてください。

コマンド・カウンタの単位はパルスです。カウンタ値はカウンタ・クリア信号または原点機能でリセットできます。また、ソフトウェアのコマンド・カウンタ設定機能を使ってリセットすることもできます。

4.5.2 フィードバック位置カウンタ

フィードバック位置カウンタは 28-bit バイナリのアップ / ダウン・カウンタです。その入力ソースは EA/EB ピンからの入力パルスです。フィードバック位置カウンタはモータのエンコーダ出力から

のモータ位置をカウントします。外部のエンコーダ入力がない場合、このカウンタはオプションでコマンド位置のソースから設定できます。

コマンド出力パルスとフィードバック入力パルスは必ずしもミニメータで同じ比率とはなりません。それら 2 つのパルスが 1:1 ではない場合は比率を設定してください。

ADLINK のモーション・コントローラは閉ループ式ではないので、モーション稼働後、フィードバック位置カウンタは参照用に使用されるだけです。位置の閉ループはサーボ・モータ・ドライバによって実行されます。サーボ・ドライバが適切に調整されており、機械部品がよく組み合わされている場合、モーション・コマンド完了後、全体の位置エラーは受け入れ可能な範囲にとどまります。

4.5.3 コマンドおよびフィードバック・エラー・カウンタ

コマンドおよびフィードバック・エラー・カウンタはコマンド位置とフィードバック位置の間のエラーを計算するのに使用されます。値はフィードバック位置を差し引いて、コマンドから計算されます。

コマンドとフィードバックの比率が 1:1 ではない場合、エラー・カウンタに意味はありません。

このカウンタは 16-bit バイナリのアップ / ダウン・カウンタです。

4.5.4 汎用カウンタ

以下のどれも汎用カウンタのソースとなり得ます :

1. コマンド位置出力 - コマンド位置カウンタと同じ
2. フィードバック位置入力 - フィードバック位置カウンタと同じ
3. 手動パルス入力 - デフォルト設定
4. クロック・チック - タイマのカウンタ、約 9.8MHz

4.5.5 ターゲット位置レコーダ

ターゲット位置レコーダはターゲット位置情報を提供するのに使用されます。また、モーション・コントローラは現在のコマンドの相対パルスを計算してから結果をプリレジスタに送信するために直前のモーション・コマンドのターゲット位置と現在のモー

ション・コマンドのターゲット位置を知る必要があるので、ターゲット位置レコーダは連続モーションで使用されます。連続モーションの前に、ターゲット位置が現在のコマンド位置と同じかどうかを確認してください。特に、原点機能や停止機能の実行後はそうしてください。

4.6 コンパレータ

各軸には 5 つのカウンタ・コンパレータがあります。コンパレータには以下のような専用機能を備えています :

1. コマンド・カウンタの正ソフト・エンド・リミット・コンパレータ
2. コマンド・カウンタの負ソフト・エンド・リミット・コンパレータ
3. コマンドおよびフィードバック・エラー・カウンタのコンパレータ
4. すべてのカウンタの汎用コンパレータ
5. すべてのコマンドおよびフィードバック・カウンタのトリガ・コンパレータ

4.6.1 ソフト・エンド・リミット・コンパレータ

各軸のエンド・リミット機能には 2 つのコンパレータがあります。ADLINK ではそれらをソフト・・エンド・リミット・コンパレータを呼んでいます。1 つはプラスすなわち正のエンド・リミットに、もう 1 つはマイナスすなわち負のエンド・リミットに使用されます。エンド・リミットは動きすぎたマシンの損傷を防ぐために使用されます。実際のエンド・リミット・スイッチの代わりにソフト・リミットを使用できます。上記の 2 つのコンパレータはコマンド位置のカウンタを比較するだけであることに注意してください。コマンド位置が正または負のコンパレータ内部の制限設定を超えると、エンド・リミット・スイッチに接触した場合と同じように稼働を停止します。

4.6.2 コマンドおよびフィードバック・エラー・カウンタのコンパレータ

このコンパレータはコマンドおよびフィードバック・カウンタ・エラーにのみ使用されます。このコンパレータを使えば、エラー

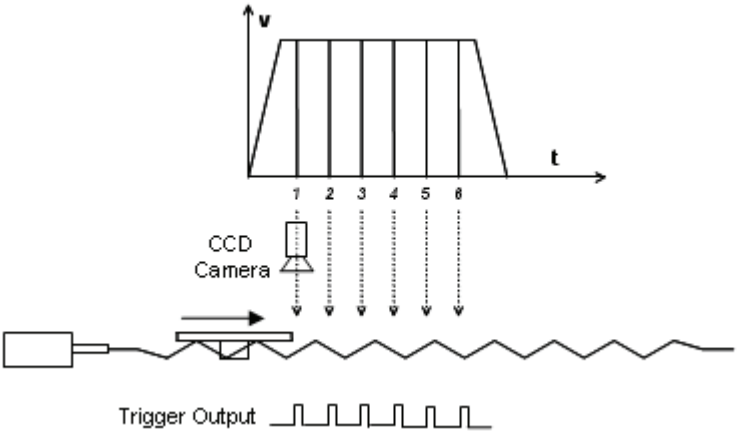
の深刻さを確認できます。また、条件に合致した場合のアクションを設定できます。アクションには、割り込み発生、直ちに停止、減速して停止などが含まれます。

4.6.3 汎用コンパレータ

汎用コンパレータを使えば、比較するソースを選択できます。ソースはコマンド、フィードバック位置カウンタ、エラー・カウンタ、または汎用カウンタから選択できます。比較方法は、方向を含む、または含まない、イコール、以上、以下から選択できます。また、条件合致時のアクションは割り込み発生や停止モーションなどから選択できます。

4.6.4 トリガ・コンパレータ

トリガ・コンパレータは汎用コンパレータとよく似ています。トリガ・コンパレータは条件合致時にトリガ・パルスを発生させる機能が追加されています。条件が合致すると、コネクタの CMP ピンはカメラに写真をとらせるように特定目的のパルスを出力します。すべての軸がこの機能を有しているとは限りません。軸に CMP ピンがあるかどうかによります。下図はトリガの例です。



このアプリケーションでは、テーブルはモーション・コマンドで制御され、CCD カメラは CMP ピンで制御されます。比較位置に達すると、パルスが出力され、画像がキャプチャされます。これはオン・ザ・フライによる画像のキャプチャです。モーション軌

道にある画像をさらに取得したい場合は、直前の画像のキャプチャ直後に新しい比較点を設定してください。この方法で連続画像のキャプチャも可能となります。

4.7 その他のモーション機能

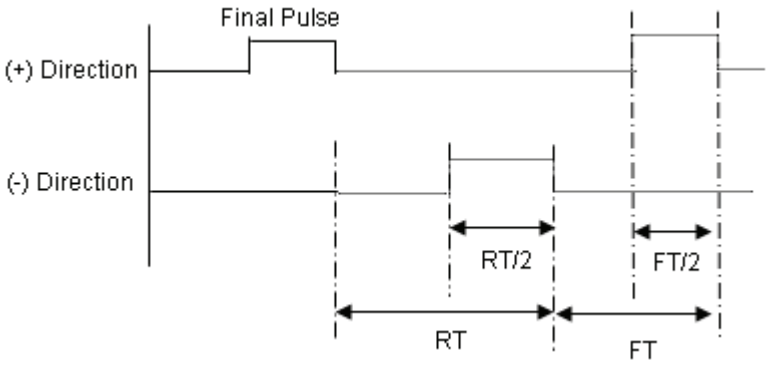
ADLINK のモーション・コントローラは上記の機能以外にも、バックラッシュ補正、スリップ補正、振動抑制、速度プロファイルの計算といった多数の機能を備えています。以下のセクションではそれらの機能について説明します。

4.7.1 バックラッシュ補正およびスリップ補正

モーション・コントローラはバックラッシュ補正およびスリップ補正機能を備えています。上記機能は FA 速度のコマンド・パルスの数を出力します。バックラッシュ補正は動作中方向が変更されるたびに実行されます。スリップ補正は方向に関係なく、モーション・コマンドの前に実行されます。パルスの補正量は関数ライブラリから設定できます。

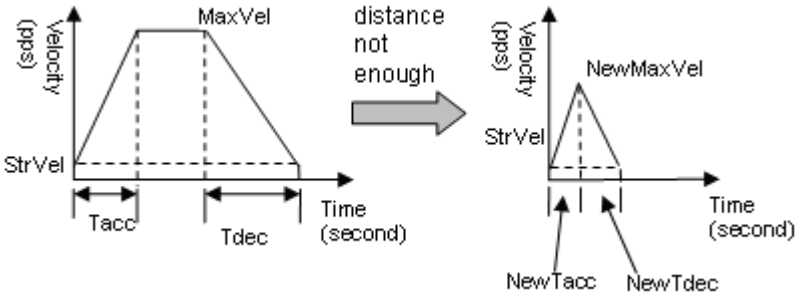
4.7.2 振動抑制機能

モーション・コントローラの振動抑制は、モーション・コマンド完了直後に、反対方向のパルスを 1 つ追加してから、前進方向のパルスを 1 つ追加する方法によって行われます。それら 2 つのダミー・パルスのタイミングは以下の図の通りです：（RT は逆進時間、FT は前進時間）



4.7.3 速度プロファイルの計算機能

ADLINK のモーション関数ではユーザーが幾つかの速度パラメータを設定しなければなりません。速度プロファイルと衝突するパラメータもあります。例えば、モーション関数に非常に高速なプロファイルと非常に短い距離を入力する場合、それらのパラメータに対応した速度プロファイルは存在しません。その場合、モーション・ライブラリは加速および減速率を維持し、ユーザーからの最大速度を下げて、実行可能な速度プロファイルを自動的に再構成します。そのしくみは下図の通りです。



ADLINK のモーション・ライブラリには、ユーザーからのコマンドの実際の速度プロファイルを検出するための一連の関数が用意されています。

4.8 割り込み制御

モーション・コントローラはホスト PC に割り込み信号を出力できます。イベント・トリガのソフトウェア・アプリケーションではそれは非常に役立ちます。この関数 _8158_int_control() を使えば、割り込みサービスを使用するかどうかを設定できます。

PCI-8158 には、モーション割り込みソース、エラー割り込みソース、GPIO 割り込みソースの 3 種類の割り込みソースがあります。モーション割り込みソースと GPIO 割り込みソースはマスカブルですが、エラー割り込みソースはマスカブルではありません。モーション割り込みソースは _8158_set_motion_int_factor() でマスカブルとなります。そのマスク・ビットは以下の表の通りです :

モーション割り込みソースのビット設定

ビット	説明
0	常時停止
1	バッファ内の次のコマンドを開始
2	コマンド・プリレジスタ 2 はエンプティで、次のコマンドの書き込みが可能
3	0
4	加速開始
5	加速終了
6	減速開始
7	減速終了
8	+ ソフト・リミットまたはコンパレータ 1 がオン
9	- ソフト・リミットまたはコンパレータ 2 がオン
10	エラー・コンパレータまたはコンパレータ 3 がオン
11	汎用コンパレータまたはコンパレータ 4 がオン
12	トリガ・コンパレータまたはコンパレータ 5 がオン
13	CLR 入力でカウンタをリセット
14	LTC 入力でカウンタをラッチ
15	ORG 入力でカウンタをラッチ
16	SD 入力がオン
17	0
18	0
19	CSTA 入力または _8158_start_move_all() がオン
20-31	0

Table 4-1: モーション割り込みソースのビット設定

エラー割り込みソースはマスク不可ですが、タイムアウトにならない限り、状況のエラー数は _8158_wait_error_interrupt() の応答コードから取得できます。

エラー割り込みの応答コード

値	説明
0	+ ソフト・リミットがオン、軸停止
1	- ソフト・リミットがオン、軸停止
2	コンパレータ 3 がオン、軸停止
3	汎用コンパレータまたはコンパレータ 4 がオン、軸停止
4	トリガ・コンパレータまたはコンパレータ 5 がオン、軸停止
5	+ エンド・リミットがオン、軸停止
6	- エンド・リミットがオン、軸停止
7	ALM 発行、軸停止
8	CSTP がオンまたは _8158_stop_move_all がオン、軸停止
9	CEMG がオン、軸停止
10	SD 入力オン、軸は減速して停止
11	0
12	補間動作エラー、停止
13	軸は他の軸のエラー停止で停止
14	パルス入力バッファがオーバーフローで停止
15	補間カウンタのオーバーフロー
16	エンコーダ入力信号エラーでも、軸は停止しない
17	パルス入力信号エラーでも、軸は停止しない
11-31	0

Table 4-2: エラー割り込みの応答コード

GPIO 割り込みソースはマスカブルです。マスク・ビットは下図の通りです :

GPIO 割り込みソースのビット設定（1= 有効、0= 無効）

ビット	説明
0	DI0 のフォーリング・エッジ
1	DI1 のフォーリング・エッジ
2	DI2 のフォーリング・エッジ
3	DI3 のフォーリング・エッジ
4	DI0 のライジング・エッジ
5	DI1 のライジング・エッジ
6	DI2 のライジング・エッジ
7	DI3 のライジング・エッジ
8	ピン 23 の入力割り込み
9	ピン 57 の入力割り込み
10	ピン 23/57 の割り込みボード指定（0= フォーリング、1= ライジング）
11-14	0
15	GPIO の割り込みスイッチ（常時 =1）

Table 4-3: GPIO 割り込みソースのビット設定

割り込み使用の手順は以下の通りです :

1. _8158_int_control(CARD_ID, 有効=1/無効=0) を使用します;
2. イベントまたは GPIO 割り込みの割り込みソースを設定します。
3. _8158_set_motion_int_facor(AXIS0, 0x01); // Axis0 は常時停止
4. _8158_set_gpio_int_factor(CARD0, 0x01); // DI0 のフォーリング・エッジ
5. _8158_wait_motion_interrupt(AXIS0, 0x01, 1000) // 常時停止割り込みを 1000ms 待機
6. _8158_wait_gpio_interrupt(CARD0, 0x01, 1000) // DI0 のフォーリング・エッジ割り込みを 1000ms 待機
7. I16 ErrNo=_8158_wait_error_interrupt(AXIS0, 2000); // エラー割り込みを 2000ms 待機

4.9　マルチ・カード動作

モーション・コントローラは 1 つのシステムで複数のカードを使用可能にします。モーション・コントローラは Plug & Play 対応なので、カードのベース・アドレスと IRQ 設定はシステム起動時に PCI BIOS によって自動的に割り当てられます。リソースの設定や設定の変更は不要です。

1 つのシステムで複数のカードを使用している場合、カードの番号に注意しなければなりません。カード番号はボードのカード ID のスイッチ設定に依存しています。軸番号はカード ID に依存します。例えば、PCI スロットに 3 枚のモーション・コントローラ・カードが接続されており、対応するカード ID が設定されている場合、それぞれのカードの軸番号は以下のようになります :

Axis / No. / Card ID	X	Y	Z	U	A	B	C	D
0	0	1	2	3	4	5	6	7
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31

複数のカードのカード ID が同じ場合は正しく機能しません。

5 MotionCreatorPro

ハードウェアの取り付け（2 章および 3 章）後、すべてのカードを正しく設定して、実行の前に、システムをダブルチェックする必要があります。この章では、制御システムを構築し、8158 カードを手動でテストして正しい動作を検証するためのガイドラインを提供します。MotionCreatorPro ソフトウェアは、8158 カードを使用するモーション制御システムのセットアップ、設定、テスト、デバッグのためのシンプルで強力な手段を提供します。

MotionCreatorPro はスクリーン解像度が 1024x768 以上の Windows 2000/ XP でのみ使用できます。推奨スクリーン解像度は 1024x768 です。DOS 環境では使用できません。

5.1 MotionCreatorPro の実行

8158 用ソフトウェア・ドライバを Windows 2000/ XP にインストールすると、MotionCreatorPro のプログラムは＜指定したパス＞¥PCI-Motion¥MotionCreatorPro にコピーされます。プログラムを実行するには、実行可能ファイルをダブルクリックするか、スタート ＞ プログラム ＞PCI-Motion＞MotionCreatorPro を使用してください。

5.2 MotionCreatorPro について

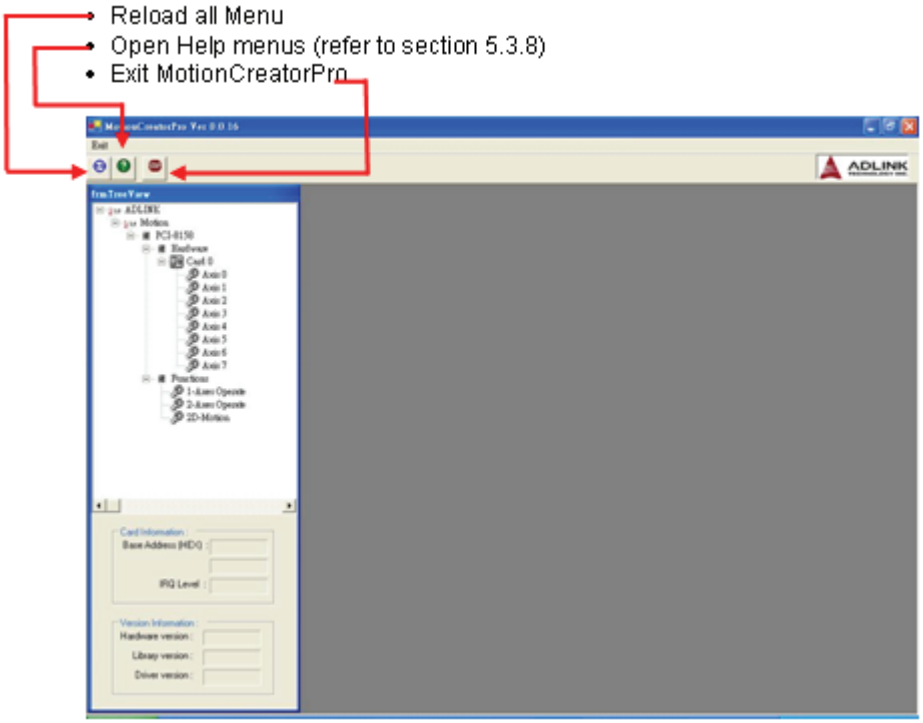
MotionCreatorPro を実行する前に、以下の問題に留意してください。

1. MotionCreatorPro は VB.NET 2003 で作成されたプログラムなので、スクリーン解像度が 1024x768 以上の Windows 2000/XP でのみ使用できます。DOS 上では使用できません。
2. 2.MotionCreatorPro を使えば、8158 カードの設定や構成を保存できます。保存された設定は MotionCreatorPro の次回実行時に自動的にロードされます。Windows ルート・ディレクトリ内の 8158.ini と 8158MC.ini の 2 つのファイルはすべての設定および構成を保存するのに使用されます。
3. 他のシステムに設定をコピーするには、8158.ini と 8158MC.ini を Windows のルート・ディレクトリにコピーしてください。
4. MotionCreatorPro が保存した同じ設定ファイルを複数の 8158 カードが使用する場合、ユーザー開発プログラムに DLL 関数呼び出し _8158_config_from_file() を呼び出すことができます。この関数は DOS 環境でも使用できます。

5.3 MotionCreatorPro の形式の紹介

5.3.1 メイン・メニュー

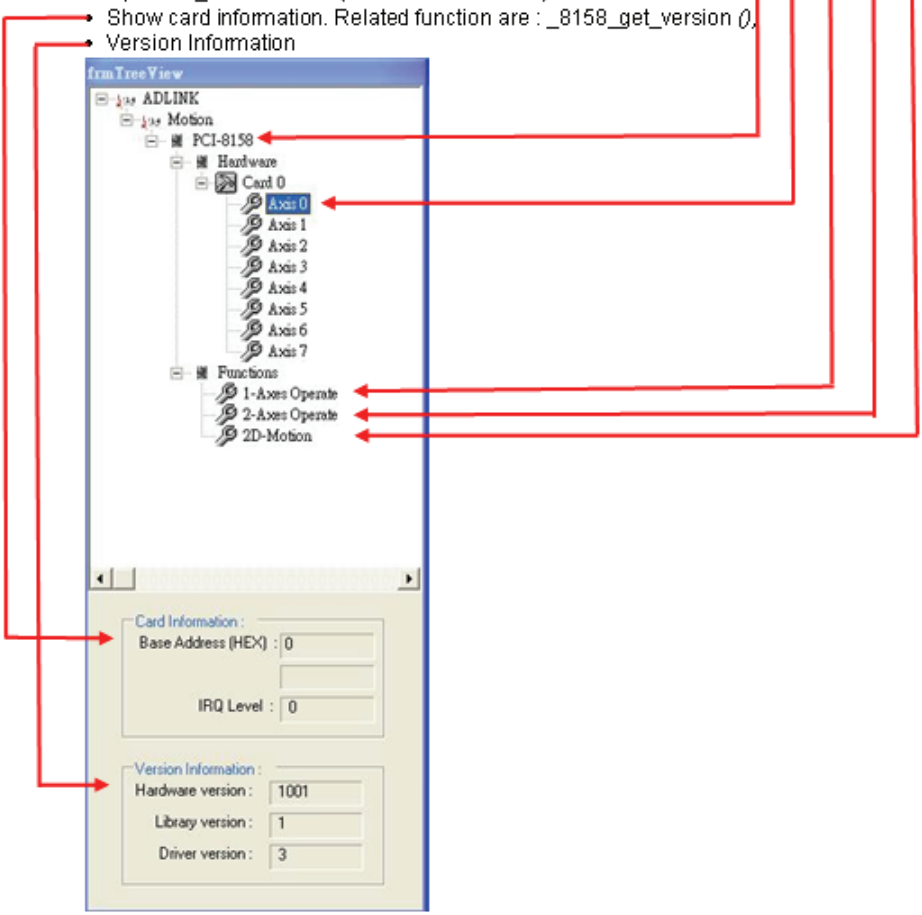
MotionCreatorPro を実行すると、メイン・メニューが表示されます。選択メニューの使用方法は以下の通りです :



5.3.2 選択メニュー

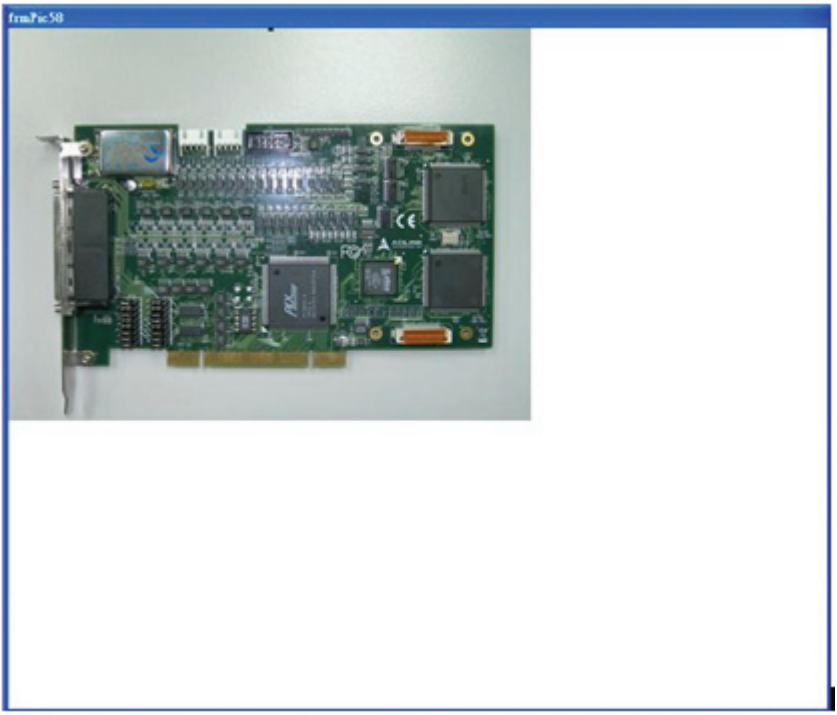
MotionCreatorPro を実行すると、選択メニューが表示されます。選択メニューの使用方法は以下の通りです :

- Select operating card and axis
- Open Card Information Menu (refer to section 5.3.3)
- Open Configuration Menu (refer to section 5.3.4)
- Open Single Axis Operation Menu (refer to section 5.3.5)
- Open Two-Axis Operation Menu (refer to section 5.3.6)
- Open 2D_Motion Menu (refer to section 5.3.7)
- Show card information. Related function are : _8158_get_version ();
- Version Information



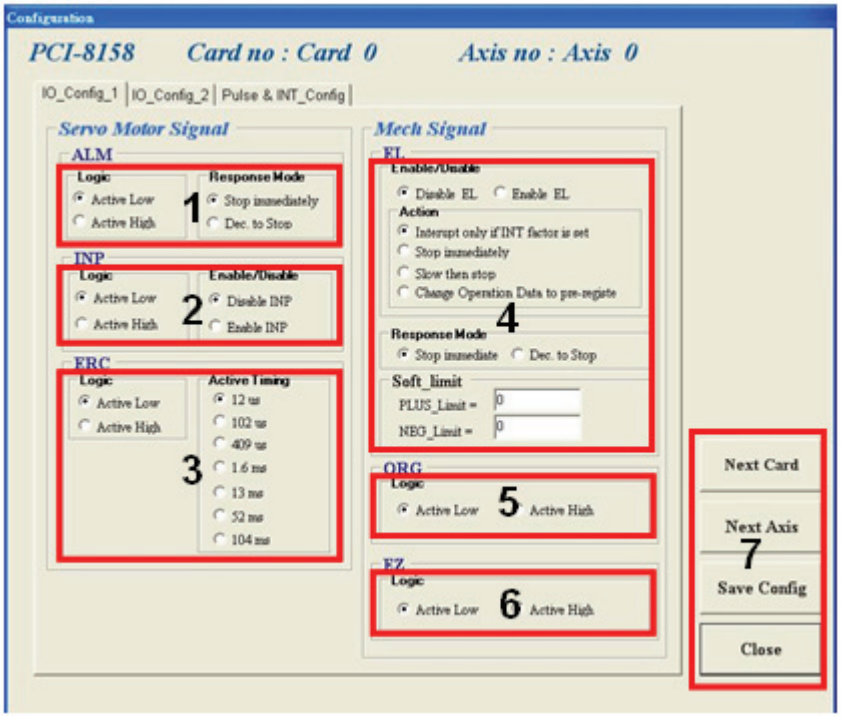
5.3.3　カード情報メニュー

このメニューはこのカードに関する情報を表示します。



5.3.4 設定メニュー

このメニューでは、ALM、INP、ERC、EL、ORG、EZ を設定できます。

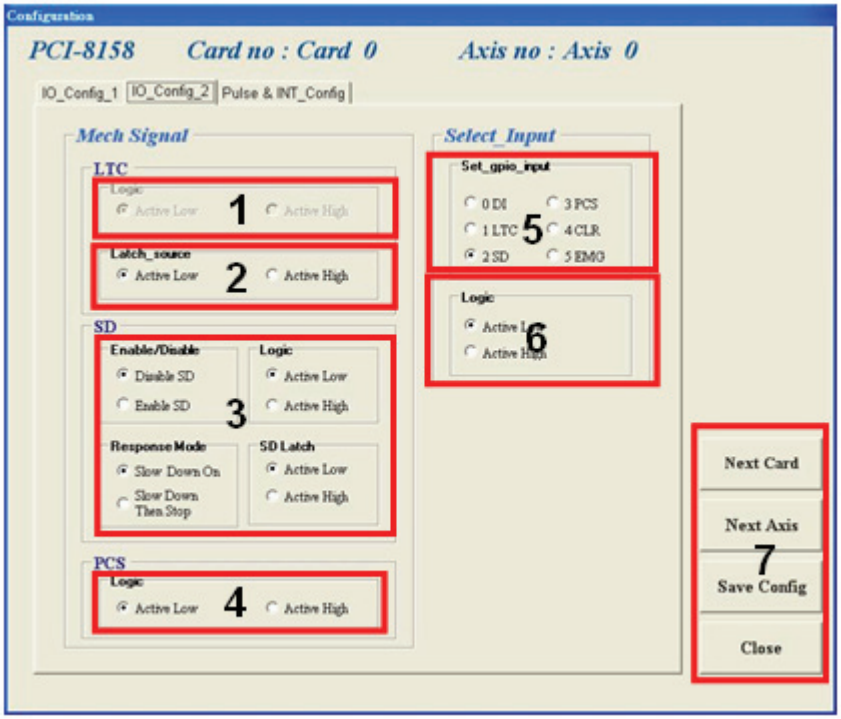


1. **ALM のロジックおよび応答モード** : ALM 信号のロジックと応答モードを選択します。関係する関数呼び出しは _8158_set_alm() です。
2. **INP のロジックとオン / オフの選択** : ロジックを選択し、INP 信号を使用するかどうかを指定します。関係する関数呼び出しは _8158_set_inp() です。
3. **ERC のロジックとアクティブ・タイミング** : ERC 信号のロジックとアクティブ・タイミングを選択します。関係する関数呼び出しは _8158_set_erc() です。
4. **EL の応答モード** : EL 信号の応答モードを選択します。関係する関数呼び出しは _8158_set_limit_logic() です。
5. **ORG のロジック** : ORG 信号のロジックを選択します。関

係する関数呼び出しは _8158_set_home_config() です。

6. **EZ のロジック**：EZ 信号のロジックを選択します。関係する関数呼び出しは _8158_set_home_config() です。
7. **ボタン**：
   - ▷ **Next Card（次のカード）**：操作するカードを変更します。
   - ▷ **Next Axis（次の軸）**：操作する軸を変更します。
   - ▷ **Save Config（設定を保存）**：現在の設定を 8158.ini および 8158MC.ini に保存します。
   - ▷ **Close（閉じる）**：メニューを閉じます。

このメニューでは、LTC、SD、PCS、Select_Input を設定できます。



1. **LTC のロジック** : LTC 信号のロジックを選択します。関係する関数呼び出しは _8158_set_ltc_logic() です。
2. **LTC の latch_source**: latch_source 信号のロジックを選択します。関係する関数呼び出しは _8158_set_latch_source () です。
3. **SD 設定** : SD 信号を設定します。関係する関数呼び出しは _8158_set_sd() です。
4. **PCSのロジック**: SelectNo信号のロジックを選択します。関係する関数呼び出しは _8158_set_pcs_logic() です。
5. **Gpio 入力の設定** : Gpio 入力の設定を選択します。関係する関数呼び出しは _8158_set_gpio_input_function です。
6. **Gpio のロジック** : Gpio のロジックを選択します。関係する関数呼び出しは _8158_set_gpio_input_function です。

7. **ボタン**：

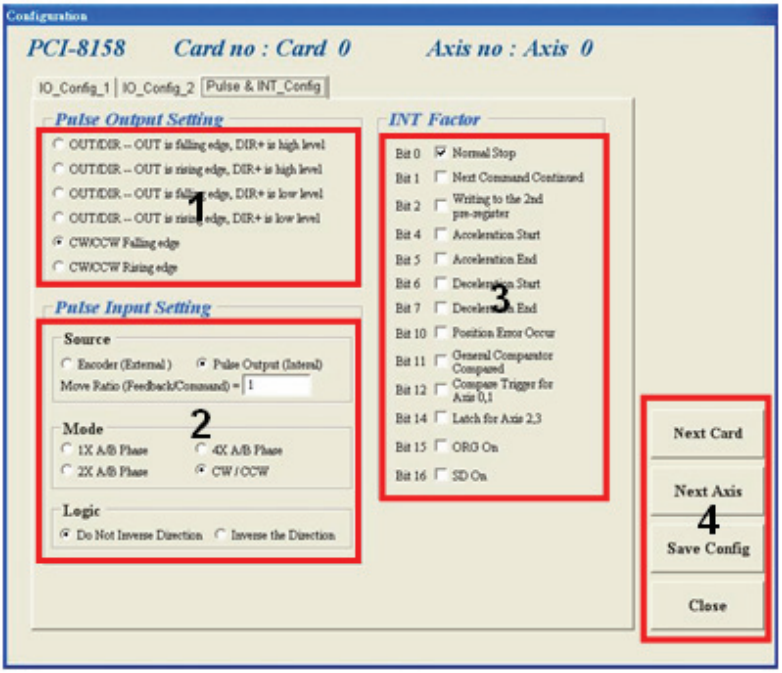
   ▷ **Next Card（次のカード）**：操作するカードを変更します。

   ▷ **Next Axis（次の軸）**：操作する軸を変更します。

   ▷ **Save Config（設定を保存）**：現在の設定を 8158.ini および 8158MC.ini に保存します。

   ▷ Close（閉じる）：メニューを閉じます。

このメニューでは、パルス入力 / 出力、移動比、INT ファクタを設定します。



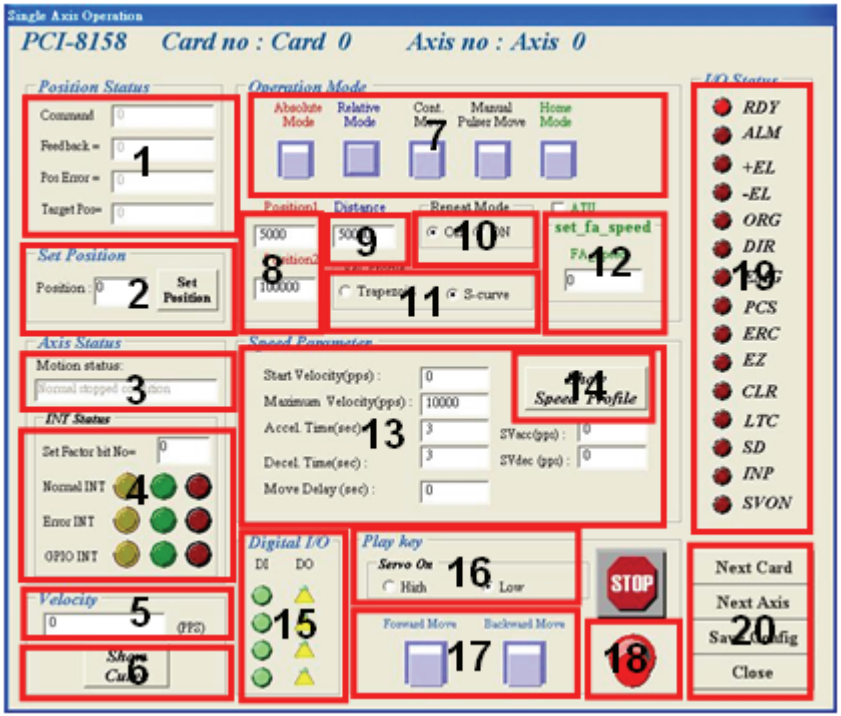
1. **パルス出力モード**：パルス信号（OUT/DIR）の出力モードを選択します。関係する関数呼び出しは _8158_set_pls_outmode() です。
2. **パルス入力**：パルス入力信号（EA/EB）を設定します。関係する関数呼び出しは _8158_set_pls_iptmode(), _8158_set_feedback_src() です。
3. **INT ファクタ**：イベント割り込みを発生させるファクタを選択します。関係する関数呼び出しは _8158_set_int_factor() です。
4. **ボタン**：
   - ▷ **Next Card（次のカード）**：操作するカードを変更します。
   - ▷ **Next Axis（次の軸）**：操作する軸を変更します。
   - ▷ **Save Config（設定を保存）**：現在の設定を 8158.ini および

8158MC.ini に保存します。

▷ **Close（閉じる）**：メニューを閉じます。

5.3.5 Single Axis Operation（1 軸動作）メニュー

このメニューでは、選択した軸の速度モード・モーション、プリセットされた相対 / 絶対モーション、手動パルス移動、原点復帰などの設定を変更できます。



1. **位置**：

   ▷ **Command（コマンド）**：コマンド・カウンタの値を表示します。関係する関数は _8158_get_command() です。

   ▷ **Feedback（フィードバック）**：フィードバック位置カウンタの値を表示します。関係する関数は _8158_get_position() です。

   ▷ **Pos Error（位置エラー）**：位置エラー・カウンタの値を表示します。関係する関数は _8158_get_error_counter() です。

   ▷ **Target Pos（ターゲット位置）**：ターゲット位置レコー

ダの値を表示します。関係する関数は _8158_get_target_pos() です。

2. **位置のリセット**：このボタンをクリックすると、すべての位置決めカウンタが特定の値に設定されます。関係する関数は以下の通りです：

   _8158_set_position()
   _8158_set_command()
   _8158_reset_error_counter()
   _8158_reset_target_pos()

3. **モーション・ステータス**：_8158_motion_done 関数の応答値を表示します。関係する関数は _8158_motion_done() です。

4. **INT ステータス**：

   ▷ **int_factor bit no（int_factor のビット数）**：int_factor のビットを設定します。

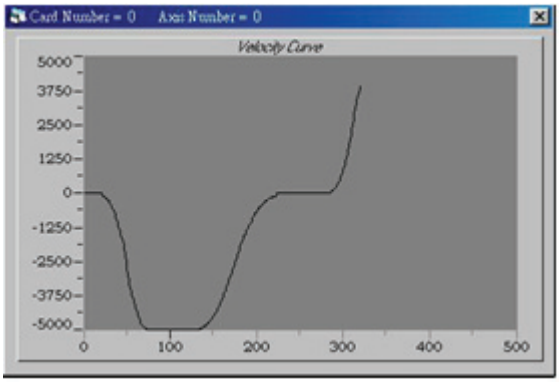
   ▷ **Normal INT（通常 INT）**：通常時の INT ステータスを表示します。関係する関数は _8158_wait_motion_interrupt () です。

   ▷ **Error INT（エラー INT）**：エラー時の INT ステータスを表示します。関係する関数は _8158_wait_error_interrupt() です。

   ▷ **GPIO INT**: GPIO の INT ステータスを表示します。関係する関数は _8158_wait_error_interrupt() です。

5. **Velocity（速度）**：PPS 単位の速度の絶対値。関係する関数は _8158_get_current_speed() です。

6. **速度曲線表示ボタン**：このボタンをクリックすると、速度と時間の曲線を表示するウィンドウが開きます。この曲線では、100ms ごとに新しい速度データ点が追加されます。ウィンドウを閉じるには、同じボタンを再度クリックします。データをクリアするには、曲線上をクリックします。



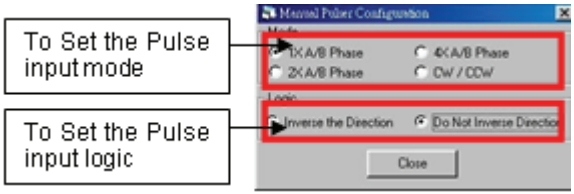
7. **Operation Mode（動作モード）**: 動作モードを選択します。

   ▷ **Absolute Mode（絶対モード）**：「Position1（位置 1）」と「Position2（位置 2）」はモーションの絶対テーゲット位置に使用されます。関係する関数は _8158_start_ta_move(), _8158_start_sa_move() です。

   ▷ **Relative Mode（相対モード）**：「Distance（距離）」はモーションの相対変位に使用されます。関係する関数は _8158_start_tr_move(), _8158_start_sr_move() です。

   ▷ **Cont. Move（連続移動）**：速度のモーション・モード。関係する関数は _8158_tv_move(), _8158_start_sv_move() です。

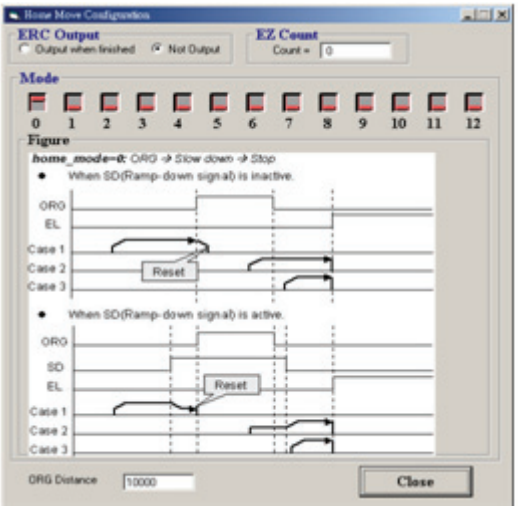
   ▷ **Manual Pulse Move（手動パルス・モーション）**：手動パルス・モーション。このボタンをクリックすると、手動パルス設定のウィンドウが開きます。



   ▷ **Home Mode（原点モード）**：原点復帰モーション。このボタンをクリックすると、原点移動設定のウィンドウが開きます。関係する関数は _8158_set_home_config() で

す。「ATU」のチェックボックスをオンにすると、モーション開始時に自動原点復帰を実行します。

- ▷ **ERC Output（ERC 出力）**：原点移動完了時、ERC 信号を送信するかどうかを指定します。
- ▷ **EZ Count（EZ カウント）**：特定の原点復帰モードで有効な EZ カウント数を設定します。
- ▷ **Mode（モード）**：原点復帰モードを選択します。13 のモードが用意されています。
- ▷ **Home Mode figure（原点モード図）**：個々の原点モードの動作を説明する図が表示されます。
- ▷ **Close（閉じる）**：このボタンをクリックすると、このウィンドウが閉じます。



8. **Position（位置）**：「Absolute Mode（絶対モード）」の絶対位置を設定します。これは「Absolute Mode（絶対モード）」選択時のみ有効です。
9. **Distance（距離）**：「Relative Mode（相対モード）」の相対距離を設定します。これは「Relative Mode（相対モード）」選択時のみ有効です。
10. **Repeat Mode（繰り返しモード）**：「On（オン）」を選択すると、モーションは繰り返しモード（前進 <-> 逆進ま

たは位置 1<-> 位置 2）になります。これは「Relative Mode（相対モード）」または「Absolute Mode（絶対モード）」選択時のみ有効です。

11. **Vel. Profile（速度プロファイル）**：速度プロファイルを選択します。「Absolute Mode（絶対モード）」、「Relative Mode（相対モード）」、「Cont. Move（連続移動）」では Trapezoidal（台形）と S-Curve（S 曲線）が使用できます。

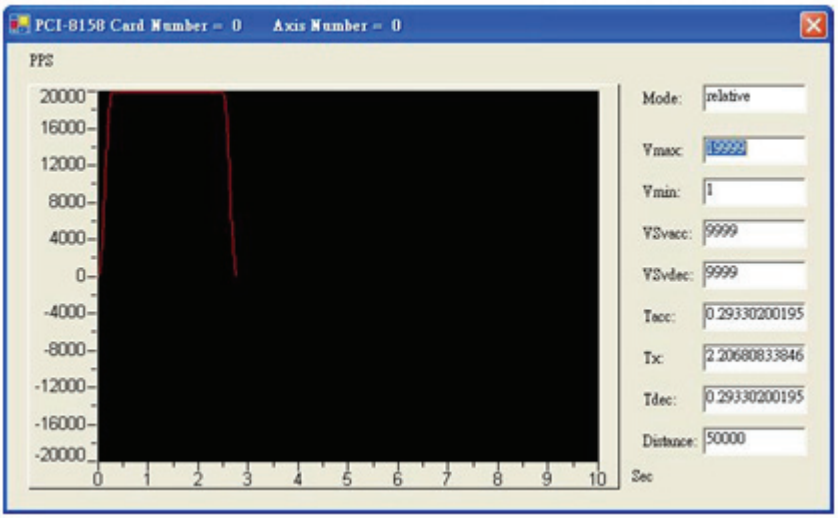
12. **FA Speed/ATU（FA 速度 /ATU）**：FA 速度を設定します。関係する関数呼び出しは _8158_set_fa_speed() です。「ATU」のチェックボックスをオンにすると、モーション開始時に自動原点復帰を実行します。

13. **Motion Parameters（モーション・パラメータ）**：1 軸モーションのパラメータを設定します。速度と移動距離はパルス入力によって決定されるので、「Manual Pulse Move（手動パルス移動）」が選択されている場合、このパラメータは無効になります。

    - ▷ **Start Velocity（初速度）**：モーションの初速を PPS 単位で設定します。値は「Absolute Mode（絶対モード）」または「Relative Mode（相対モード）」でのみ有効です。例えば、-100.0 は 100.0 と同じです。「Cont. Move（連続移動）」では、値と符号の両方が有効になります。-100.0 はマイナス方向の 100.0 を意味します。
    - ▷ **Maximum Velocity（最大速度）**：モーションの最大速度を PPS 単位で設定します。値は「Absolute Mode（絶対モード）」または「Relative Mode（相対モード）」でのみ有効です。例えば、-5000.0 は 5000.0 と同じです。「Cont. Move（連続移動）」では、値と符号の両方が有効になります。-5000.0 はマイナス方向の 5000.0 を意味します。
    - ▷ **Accel. Time（加速時間）**：加速時間を秒単位で設定します。
    - ▷ **Decel. Time（減速時間）**：減速時間を秒単位で設定します。
    - ▷ **SVacc**: 加速時の S 曲線の範囲を PPS 単位で設定します。
    - ▷ **SVdec**: 減速時の S 曲線の範囲を PPS 単位で設定します。
    - ▷ **Move Delay（移動遅延）**：この設定は繰り返しモードが

「On（オン）」に設定されている場合のみ有効です。その場合、8158 は次のモーションに移る前に、指定時間待機します。

14. **Speed_Profile**:　このボタンをクリックすると、速度プロファイルが表示されます。



15. **Digital I/O（デジタル I/O）**：デジタル I/O の表示と設定。関係する関数は _8158_get_gpio_output(),_8158_get_gpio_input(), _8158_set_gpio_output() です。

16. **Servo On（サーボ・オン）**：SVON 信号の出力ステータスを設定します。関係する関数は _8158_set_servo() です。

17. **Play Key（再生キー）**：

左再生キー: このボタンをクリックすると、8158 は直前の設定に従ってパルスの出力を開始します。

- ▷ 「Absolute Mode（絶対モード）」では、軸が position1（位置 1）に移動します。
- ▷ 「Relative Mode（相対モード）」では、軸が前進します。
- ▷ 「Cont. Move（連続移動）」では、軸は速度設定に従って移動を開始します。
- ▷ 「Manual Pulse Move（手動パルス移動）」では、軸はパルス移動に入ります。速度制限は「Maximum Velocity（最大速度）」で設定された値です。

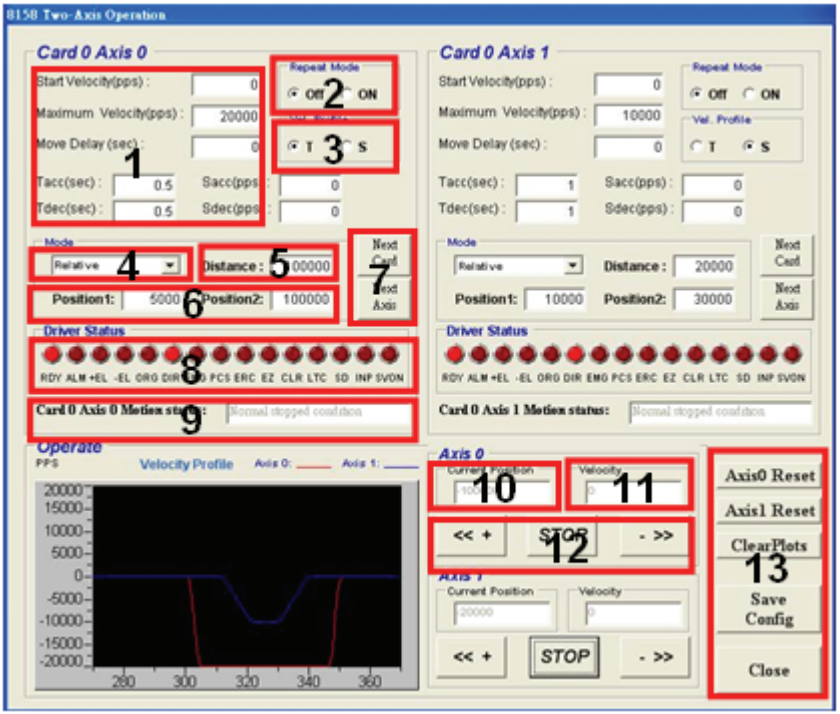
右再生キー: このボタンをクリックすると、8158 は直前の設定に従ってパルスの出力を開始します。

- ▷ 「Absolute Mode（絶対モード）」では、軸が position1（位置 1）に移動します。
- ▷ 「Relative Mode（相対モード）」では、軸が後退します。
- ▷ 「Cont. Move（連続移動）」では、軸は速度設定に従って逆方向に移動を開始します。
- ▷ 「Manual Pulse Move（手動パルス移動）」では、軸はパルス移動に入ります。速度制限は「Maximum Velocity（最大速度）」で設定された値です。

18. **Stop（停止）ボタン**：このボタンをクリックすると、8158 は減速して停止します。減速時間は「Decel. Time（減速時間）」で定義されます。関係する関数は _8158_sd_stop() です。
19. **I/O Status（I/O ステータス）**：モーション I/O の状態。点灯はアクティブ状態、消灯は非アクティブ状態を表します。関係する関数は _8158_get_io_status() です。
20. **ボタン**：
    - ▷ **Next Card（次のカード）**：操作するカードを変更します。
    - ▷ **Next Axis（次の軸）**：操作する軸を変更します。
    - ▷ **Save Config（設定を保存）**：現在の設定を 8158.ini および 8158MC.ini に保存します。
    - ▷ **Close（閉じる）**：メニューを閉じます。

5.3.6 Two-Axis Operation（2 軸動作）メニュー

このメニューでは、2 つの選択した軸の速度モード・モーションやプリセットされた相対 / 絶対モーションなどの設定を変更できます。



1. **Motion Parameters（モーション・パラメータ）**：1 軸モーションのパラメータを設定します。速度と移動距離はパルス入力によって決定されるので、「Manual Pulse Move（手動パルス移動）」が選択されている場合、このパラメータは無効になります。

   ▷ **Start Velocity（初速度）**：モーションの初速を PPS 単位で設定します。値は「Absolute Mode（絶対モード）」または「Relative Mode（相対モード）」でのみ有効です。例えば、-100.0 は 100.0 と同じです。

   ▷ **Maximum Velocity（最大速度）**：モーションの最大速度を PPS 単位で設定します。値は「Absolute Mode（絶対

モード）」または「Relative Mode（相対モード）」でのみ有効です。例えば、-5000.0 は 5000.0 と同じです。

- ▷ **Tacc**: 加速時間を秒単位で設定します。
- ▷ **Tdec**: 減速時間を秒単位で設定します。
- ▷ **Sacc**: 加速時の S 曲線の範囲を PPS 単位で設定します。
- ▷ **Sdec**: 減速時の S 曲線の範囲を PPS 単位で設定します。

2. **Repeat Mode（繰り返しモード）**：「On（オン）」を選択すると、モーションは繰り返しモード（前進 <-> 逆進または位置 1<-> 位置 2）になります。これは「Relative Mode（相対モード）」または「Absolute Mode（絶対モード）」選択時のみ有効です。
3. **Vel. Profile（速度プロファイル）**：速度プロファイルを選択します。「Absolute Mode（絶対モード）」、「Relative Mode（相対モード）」、「Cont. Move（連続移動）」では Trapezoidal（台形）と S-Curve（S 曲線）が使用できます。
4. **Operation Mode（動作モード）**：動作モードを選択します。
   - ▷ **Absolute Mode（絶対モード）**：「Position1（位置 1）」と「Position2（位置 2）」はモーションの絶対テーゲット位置に使用されます。関係する関数は _8158_start_ta_move(), _8158_start_sa_move() です。
   - ▷ **Relative Mode（相対モード）**：「Distance（距離）」はモーションの相対変位に使用されます。関係する関数は _8158_start_tr_move(), _8158_start_sr_move() です。
5. **Distance（距離）**：「Relative Mode（相対モード）」の相対距離を設定します。これは「Relative Mode（相対モード）」選択時のみ有効です。
6. **Position（位置）**：「Absolute Mode（絶対モード）」の絶対位置を設定します。これは「Absolute Mode（絶対モード）」選択時のみ有効です。
7. **ボタン**：
   - ▷ **Next Card（次のカード）**：操作するカードを変更します。
   - ▷ **Next Axis（次の軸）**：操作する軸を変更します。
8. **I/O Status（I/O ステータス）**：モーション I/O の状態。点

灯はアクティブ状態、消灯は非アクティブ状態を表します。関係する関数は _8158_get_io_status() です。

9. **モーション・ステータス**：_8158_motion_done 関数の応答値を表示します。関係する関数は _8158_motion_done() です。

10. **Current Position（現在位置）**：

    ▷ **Command（コマンド）**：コマンド・カウンタの値を表示します。関係する関数は _8158_get_position() です。

11. **Velocity（速度）**：PPS 単位の速度の絶対値。関係する関数は _8158_get_current_speed() です。

12. **Play Key（再生キー）**：

左再生キー：このボタンをクリックすると、8158 は直前の設定に従ってパルスの出力を開始します。

- ▷「Absolute Mode（絶対モード）」では、軸が position1（位置 1）に移動します。
- ▷「Relative Mode（相対モード）」では、軸が前進します。

右再生キー：このボタンをクリックすると、8158 は直前の設定に従ってパルスの出力を開始します。

- ▷「Absolute Mode（絶対モード）」では、軸が position2（位置 2）に移動します。
- ▷「Relative Mode（相対モード）」では、軸が後退します。

Stop（停止）ボタン：このボタンをクリックすると、8158 は減速して停止します。減速時間は「Decel. Time（減速時間）」で定義されます。関係する関数は _8158_sd_stop() です。

13. **ボタン**：

    ▷ **Axis0 Reset（Axis0 のリセット）**：このボタンをクリックすると、選択した軸のすべての位置決めカウンタがゼロに設定されます。関係する関数は以下の通りです：

```
_8158_set_position()
_8158_set_command()
_8158_reset_error_counter()
```

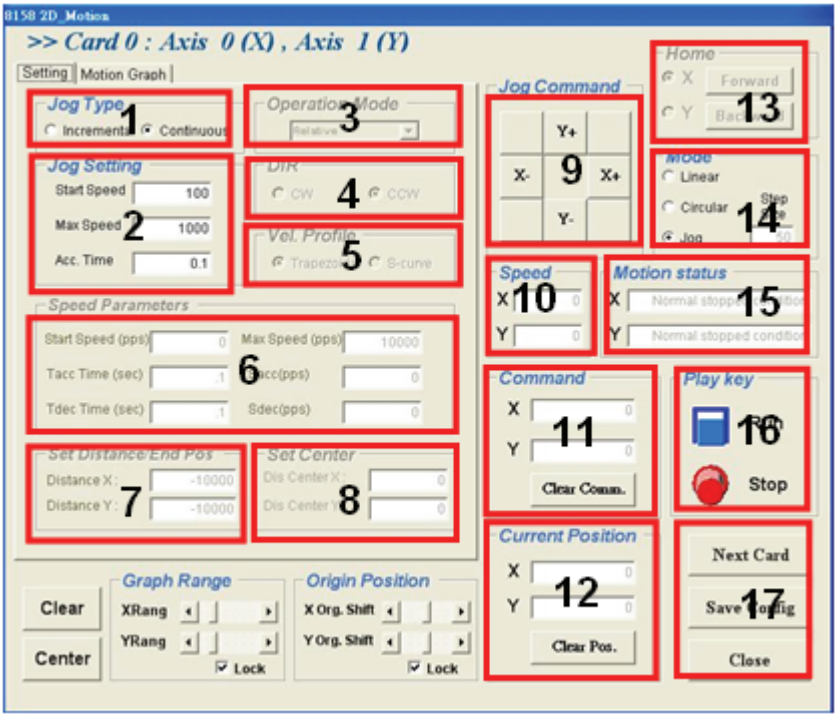
_8158_reset_target_pos()

- ▷ **Axis1 Reset（Axis1 のリセット）**：このボタンをクリックすると、選択した軸のすべての位置決めカウンタがゼロに設定されます。
- ▷ **ClearPlots**: モーション・グラフをクリアします。
- ▷ **Save Config（設定を保存）**：現在の設定を 8158.ini および 8158MC.ini に保存します。
- ▷ **Close（閉じる）**：メニューを閉じます。

5.3.7 2D_Motion メニュー

操作ウィンドウの 2-D ボタンを押すと、このウィンドウに入ります。これは 2-D モーションのテストに使用されます。内容は以下の通りです：

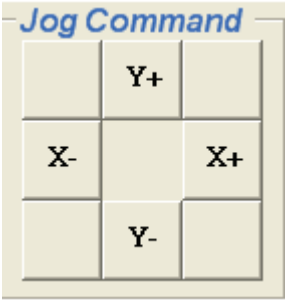
- ▶ 直線補間
- ▶ 円形補間
- ▶ インクリメンタル・ジョグ
- ▶ 連続ジョグ
- ▶ その他の制御対象



1. **Jog Type（ジョグ・タイプ）**：

   ▷ **Continuous Jog（連続ジョグ）**：連続ジョグでは、方向ボタンを押すと、軸は速度を上げて連続移動します。

押している時間が長いほど、速度が速くなります。ボタンを放すと、軸は直ちに停止します。



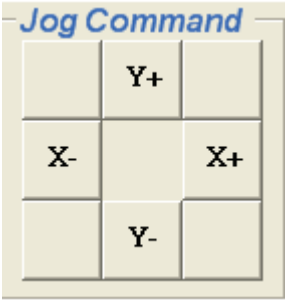
▷ **Incremental Jog（インクリメンタル・ジョグ）**：インクリメンタル・ジョグでは、方向ボタンをクリックすると、軸は Step-Size（ステップ・サイズ）設定に従って一定の距離をステップします。

2. **Jog Setting（ジョグ設定）**：1 軸モーションのパラメータを設定します。速度と移動距離はパルス入力によって決定されるので、「Jog Mode（ジョグモード）」が選択されている場合、このパラメータは無効になります。
   - ▷ **Start Velocity（初速度）**：モーションの初速を PPS 単位で設定します。
   - ▷ **Maximum Velocity（最大速度）**：モーションの最大速度を PPS 単位で設定します。
   - ▷ **Tacc**: 加速時間を秒単位で設定します。
3. **Operation Mode（動作モード）**: 動作モードを選択します。
   - ▷ **Absolute Mode（絶対モード）**：「直線補間モード」が選択されている場合、「Position（位置）」はモーションの絶対ターゲット位置に使用されます。「円形補間モード」が選択されている場合は、「ABS EndPos」と「ABS Center」がモーションの絶対ターゲット位置に使用されます。関係する関数は _8158_start_ta_move(), _8158_start_sa_move() です。

- ▷ **Relative Mode（相対モード）**：「直線補間モード」が選択されている場合、「Distance（距離）」はモーションの絶対ターゲット位置に使用されます。「円形補間モード」が選択されている場合は、「Dis EndPos」と「Dis Center」がモーションの絶対ターゲット位置に使用されます。関係する関数は _8158_start_tr_move(), _8158_start_sr_move() です。

4. **DIR**: 円弧、CW/CCW の指定した距離。「円形補間モード」選択時のみ有効です。
5. **Vel. Profile（速度プロファイル）**：速度プロファイルを選択します。「直線補間モード」と「円形補間モード」では Trapezoidal（台形）と S-Curve（S 曲線）の両方が使用できます。
6. **Motion Parameters（速度パラメータ）**：1 軸モーションのパラメータを設定します。速度と移動距離はパルス入力によって決定されるので、「直線補間モード」または「円形補間モード」が選択されている場合、このパラメータは無効になります。
   - ▷ **Start Velocity（初速度）**：モーションの初速を PPS 単位で設定します。
   - ▷ **Maximum Velocity（最大速度）**：モーションの最大速度を PPS 単位で設定します。
   - ▷ **Accel. Time（加速時間）**：加速時間を秒単位で設定します。
   - ▷ **Decel. Time（減速時間）**：減速時間を秒単位で設定します。
   - ▷ **SVacc**: 加速時の S 曲線の範囲を PPS 単位で設定します。
   - ▷ **SVdec**: 減速時の S 曲線の範囲を PPS 単位で設定します。
7. **Set Distance/End Pos（距離 / 終点位置の設定）**：「直線補間モード」では絶対ターゲット位置または相対距離を設定します。「円形補間モード」では円弧の終点位置を設定します。「直線補間モード」と「円形補間モード」で使用できます。
8. **Set Center（中心の設定）**：「円形補間モード」で中心の位置を設定します。これは「円形補間モード」選択時のみ有効です。

9. **Jog Command（ジョグ・コマンド）**：方向ボタンを押すと移動します。

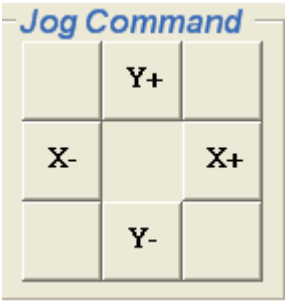


10. **Velocity（速度）**：PPS 単位の速度の絶対値。関係する関数は _8158_get_current_speed() です。
11. **Interpolation Command（補間コマンド）**：
    - ▷ Command（コマンド）：コマンド・カウンタの値を表示します。関係する関数は _8158_get_command() です。
12. **Current Position（現在位置）**：
    - ▷ Feedback（フィードバック）：フィードバック位置カウンタの値を表示します。関係する関数は _8158_get_position() です。
13. **Home Mode（原点モード）**：原点復帰モーション。このボタンをクリックすると、原点移動設定のウィンドウが開きます。関係する関数は _8158_set_home_config() です。このウィンドウの左下には 2 つの原点復帰ボタンが用意されています。これは原点に復帰する必要がある場合に役立ちます。
14. **モード**：
    - ▷ **Linear Interpolation（直線補間）**：「Motion Parameters Setting Frame（モーション・パラメータ設定フレーム）」でモーションのパラメータを正しく設定した後、このフレームに移動先を入力できます。その後、Run（実行）ボタンをクリックすると、直線補間モーションがスタートします。
    - ▷ **Circular Interpolation（円形補間）**：円形補間モードの設定では、「Motion Parameters Setting Frame（モーショ

ン・パラメータ設定フレーム)」に、円弧角度、分割軸、最適化オプションの 3 つのパラメータを追加できます。設定方法についてセクション 6.7 および 6.8 を参照してください。

上記パラメータ設定後、「Play Key Frame（再生キー・フレーム)」に円弧の中心と角度を入力できます。Run（実行）ボタンをクリックすると、円形補間モーションがスタートします。

▷ **Jog Type（ジョグ・タイプ）**: Continuous Jog（連続ジョグ）



連続ジョグでは、方向ボタンを押すと、軸は速度を上げて連続移動します。押している時間が長いほど、速度が速くなります。ボタンを放すと、軸は直ちに停止します。

▷ **Incremental Jog（インクリメンタル・ジョグ）**: インクリメンタル・ジョグでは、方向ボタンをクリックすると、軸は Step-Size（ステップ・サイズ）設定に従って一定の距離をステップします。

15. **モーション・ステータス**: _8158_motion_done 関数の応答値を表示します。関係する関数は _8158_motion_done() です。

16. **Play Key（再生キー）**:

▶ **再生ボタン**: このボタンをクリックすると、8158 は直前の設定に従ってパルスの出力を開始します。

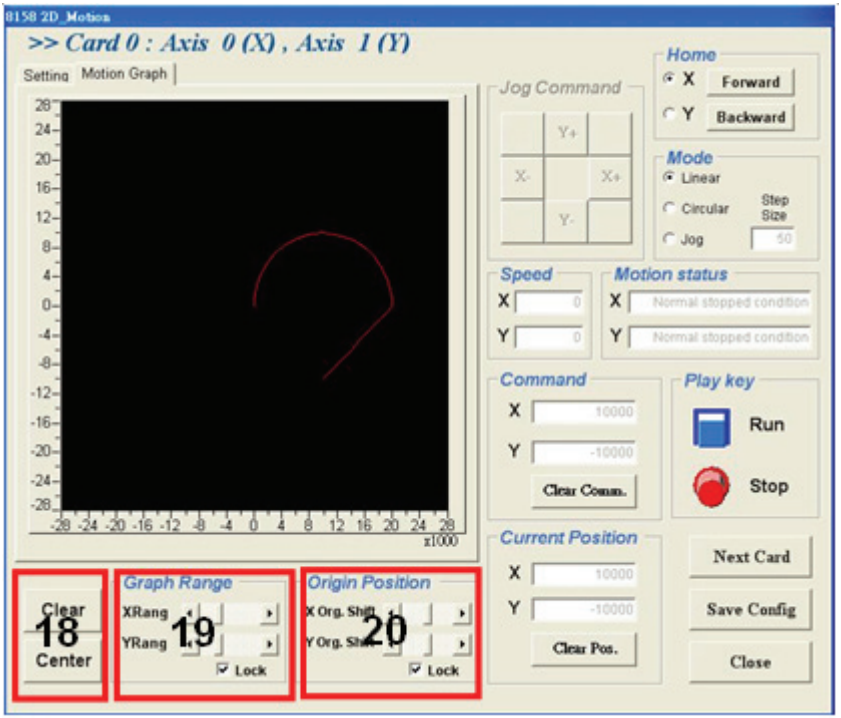
- ▷ 「Linear Mode（直線モード）」では、軸が Distance（距離）に移動します。関係する関数は _8158_start_tr_move_xy, _8158_start_sr_move_xy です。
- ▷ 「Circular Mode（円形モード）」では、軸は Distance（距離）（Pos/Dist（パルス）によって）に移動します。関係する関数は _8158_start_tr_arc_xy, _8158_start_sr_arc_xy です。

▶ **Stop（停止）ボタン**：このボタンをクリックすると、8158 は減速して停止します。減速時間は「Decel. Time（減速時間）」で定義されます。関係する関数は _8158_sd_stop() です。

17. **ボタン**：

- ▷ **Next Card（次のカード）**：操作するカードを変更します。
- ▷ **Save Config（設定を保存）**：現在の設定を 8158.ini および 8158MC.ini に保存します。

▷ **Close（閉じる）**：メニューを閉じます。



18. **Graph Range Frame（グラフ範囲フレーム）**：
    - ▷ **Clear（クリア）**：モーション・グラフをクリアします。
    - ▷ **Center（中心）**：モーション・グラフを中心位置に表示します。
19. **Graph Range（グラフ範囲）**：X または Y 軸の表示範囲を制御します。
20. **Origin Position（原点位置）**：ユーザーによる表示位置のパンを可能にします。

5.3.8 Help（ヘルプ）メニュー

このメニューでは、マウスを右クリックしてヘルプ情報を表示できます。

6　関数ライブラリ

この章ではPCI-8158カードの対応ソフトウェアについて説明します。それらの関数を使えば、C、C++、または Visual Basic でプログラムを開発できます。プログラミング環境に Delphi を使用する場合は、ヘッダ・ファイルと pci_8158.h を手動で変換する必要があります。

6.1 関数のリスト

システムおよび初期化、セクション 6.3

関数名	説明
_8158_initial	カードの初期化
_8158_close	カードの終了
_8158_get_version	ハードウェアおよびソフトウェア・バージョンの確認
_8158_set_security_key	セキュリティ・パスワードの設定
_8158_check_security_key	セキュリティ・パスワードの確認
_8158_reset_security_key	セキュリティ・パスワードをデフォルト値にリセット
_8158_config_from_file	ファイルから PCI-8158 を設定

パルス入力 / 出力設定、セクション 6.4

関数名	説明
_8158_set_pls_outmode	パルス・コマンドの出力モードを設定
_8158_set_pls_iptmode	エンコーダの入力モードを設定
_8158_set_feedback_src	カウンタの入力ソースを設定

速度モード・モーション、セクション 6.5

関数名	説明
_8158_tv_move	台形プロファイルで軸を一定の速度に加速
_8158_sv_move	S 曲線プロファイルで軸を一定の速度に加速
_8158_sd_stop	減速して、停止
_8158_emg_stop	直ちに停止
_8158_get_current_speed	現在の速度（パルス / 秒）を取得
_8158_speed_override	オン・ザ・フライで速度を変更
_8158_set_max_override_speed	最大オーバーライド速度を設定

1 軸位置モード、セクション 6.6

関数名	説明
_8158_start_tr_move	相対台形プロファイル移動を開始
_8158_start_ta_move	絶対台形プロファイル移動を開始
_8158_start_sr_move	相対 S 曲線プロファイル移動を開始
_8158_start_sa_move	絶対 S 曲線プロファイル移動を開始
_8158_set_move_ratio	コマンド・パルスとフィードバック・パルスの比を設定
_8158_position_override	オン・ザ・フライで位置を変更

直線補間モーション、セクション 6.7

関数名	説明
_8158_start_tr_move_xy	台形プロファイルで X および Y の相対 2 軸直線補間を開始
_8158_start_ta_move_xy	台形プロファイルで X および Y の絶対 2 軸直線補間を開始
_8158_start_sr_move_xy	S 曲線プロファイルで X および Y の相対 2 軸直線補間を開始
_8158_start_sa_move_xy	S 曲線プロファイルで X および Y の絶対 2 軸直線補間を開始
_8158_start_tr_move_zu	台形プロファイルで Z および U の相対 2 軸直線補間を開始
_8158_start_ta_move_zu	台形プロファイルで Z および U の絶対 2 軸直線補間を開始
_8158_start_sr_move_zu	S 曲線プロファイルで Z および U の相対 2 軸直線補間を開始
_8158_start_sa_move_zu	Z および U の S 曲線絶対円形補間を開始
_8158_start_tr_move_ab	台形プロファイルで A および B の相対 2 軸直線補間を開始
_8158_start_ta_move_ab	台形プロファイルで A および B の絶対 2 軸直線補間を開始
_8158_start_sr_move_ab	S 曲線プロファイルで A および B の相対 2 軸直線補間を開始
_8158_start_sa_move_ab	A および B の S 曲線絶対円形補間を開始
_8158_start_tr_move_cd	台形プロファイルで C および D の相対 2 軸直線補間を開始

関数名	説明
_8158_start_ta_move_cd	台形プロファイルで C および D の絶対 2 軸直線補間を開始
_8158_start_sr_move_cd	S 曲線プロファイルで C および D の相対 2 軸直線補間を開始
_8158_start_sa_move_cd	S 曲線プロファイルで C および D の絶対 2 軸直線補間を開始
_8158_start_tr_line2	4 軸の任意の 2 軸に対して台形プロファイルで相対 2 軸直線補間を開始
_8158_start_ta_line2	4 軸の任意の 2 軸に対して台形プロファイルで絶対 2 軸直線補間を開始
_8158_start_sr_line2	4 軸の任意の 2 軸に対して S 曲線プロファイルで相対 2 軸直線補間を開始
_8158_start_sa_line2	4 軸の任意の 2 軸に対して S 曲線プロファイルで絶対 2 軸直線補間を開始
_8158_start_tr_line3	4 軸の任意の 3 軸に対して台形プロファイルで相対 3 軸直線補間を開始
_8158_start_ta_line3	4 軸の任意の 3 軸に対して台形プロファイルで絶対 3 軸直線補間を開始
_8158_start_sr_line3	4 軸の任意の 3 軸に対して S 曲線プロファイルで相対 3 軸直線補間を開始
_8158_start_sa_line3	4 軸の任意の 3 軸に対して S 曲線プロファイルで絶対 3 軸直線補間を開始
_8158_start_tr_line4	4 軸の任意の 4 軸に対して台形プロファイルで相対 4 軸直線補間を開始
_8158_start_ta_line4	4 軸の任意の 4 軸に対して台形プロファイルで絶対 4 軸直線補間を開始
_8158_start_sr_line4	4 軸の任意の 4 軸に対して S 曲線プロファイルで相対 4 軸直線補間を開始
_8158_start_sa_line4	4 軸の任意の 4 軸に対して S 曲線プロファイルで絶対 4 軸直線補間を開始

円形補間モーション、セクション 6.8

関数名	説明
_8158_start_tr_arc_xy	X および Y の T 曲線相対円形補間を開始
_8158_start_ta_arc_xy	X および Y の T 曲線絶対円形補間を開始
_8158_start_sr_arc_xy	X および Y の S 曲線相対円形補間を開始

関数名	説明
_8158_start_sa_arc_xy	X および Y の S 曲線絶対円形補間を開始
_8158_start_tr_arc_zu	Z および U の T 曲線相対円形補間を開始
_8158_start_ta_arc_zu	Z および U の T 曲線絶対円形補間を開始
_8158_start_sr_arc_zu	Z および U の S 曲線相対円形補間を開始
_8158_start_sa_arc_zu	Z および U の S 曲線絶対円形補間を開始
_8158_start_tr_arc_ab	A および B の T 曲線相対円形補間を開始
_8158_start_ta_arc_ab	A および B の T 曲線絶対円形補間を開始
_8158_start_sr_arc_ab	A および B の S 曲線相対円形補間を開始
_8158_start_sa_arc_ab	A および B の S 曲線絶対円形補間を開始
_8158_start_tr_arc_cd	C および D の T 曲線相対円形補間を開始
_8158_start_ta_arc_cd	C および D の T 曲線絶対円形補間を開始
_8158_start_sr_arc_cd	C および D の S 曲線相対円形補間を開始
_8158_start_sa_arc_cd	C および D の S 曲線絶対円形補間を開始
_8158_start_tr_arc2	4 軸の任意の 2 軸に対して T 曲線相対円形補間を開始
_8158_start_ta_arc2	4 軸の任意の 2 軸に対して T 曲線絶対円形補間を開始
_8158_start_sr_arc2	4 軸の任意の 2 軸に対して S 曲線相対円形補間を開始
_8158_start_sa_arc2	4 軸の任意の 2 軸に対して S 曲線絶対円形補間を開始

原点復帰モード、セクション 6.9

関数名	説明
_8158_set_home_config	原点 / インデックス・ロジックの設定
_8158_home_move	原点復帰アクションの開始
_8158_home_search	原点自動検索を実行

手動パルス・モーション、セクション 6.10

関数名	説明
_8158_set_pulser_iptmode	パルス入力モードの設定
_8158_disable_pulser_input	パルス入力を無効にする
_8158_pulser_vmove	パルスの V 移動を開始

関数名	説明
_8158_pulser_pmove	パルスの P 移動を開始
_8158_set_pulser_ratio	手動パルス比を実際の出力パルス速度に設定

モーション・ステータス、セクション 6.11

関数名	説明
_8158_motion_done	モーション・ステータスを返す

モーション・インタフェース I/O、セクション 6.12

関数名	説明
_8158_set_servo	SVON 信号のオン / オフ状態を設定
_8158_set_pcs_logic	PCS（位置変更信号）のロジックを設定
_8158_set_pcs	PCS の位置オーバーライドを可能にする
_8158_set_clr_mode	CLR 信号のモードを設定
_8158_set_inp	INP 信号のロジックおよび動作モードを設定
_8158_set_alm	ALM 信号のロジックおよび動作モードを設定
_8158_set_erc	ERC 信号のロジックおよびタイミングを設定
_8158_set_erc_out	ERC 信号を出力
_8158_clr_erc	ERC 信号をクリア
_8158_set_sd	SD 信号のロジックおよび動作モードを設定
_8158_enable_sd	SD 信号を使用する
_8158_set_limit_logic	EL 信号のロジックを設定
_8158_set_limit_mode	EL の動作モードを設定
_8158_get_io_status	8158 のすべてのモーション I/O を取得

割り込む制御、セクション 6.13

関数名	説明
_8158_int_control	INT サービスを使用するかどうかを設定
_8158_wait_error_interrupt	エラー関連割り込みの待機
_8158_wait_motion_interrupt	モーション関連割り込みの待機
_8158_set_motion_int_factor	モーション関連割り込みのファクタを設定

ポジション制御およびカウンタ、セクション 6.14

関数名	説明
_8158_get_position	フィードバック位置カウンタの値を取得
_8158_set_position	フィードバック位置カウンタの設定
_8158_get_command	コマンド位置カウンタの値を取得
_8158_set_command	コマンド位置カウンタの設定
_8158_get_error_counter	位置エラー・カウンタの値を取得
_8158_reset_error_counter	位置エラー・カウンタをリセット
_8158_get_general_counter	汎用カウンタの値を取得
_8158_set_general_counter	汎用カウンタの設定
_8158_get_target_pos	ターゲット位置レコーダの値を取得
_8158_reset_target_pos	ターゲット位置レコーダをリセット
_8158_get_res_distance	モーションから蓄積された残存パルスを取得
_8158_set_res_distance	残存パルス記録の設定

位置比較およびラッチ、セクション 6.15

関数名	説明
_8158_set_trigger_logic	CMP 信号のロジックを設定
_8158_set_error_comparator	エラー・コンパレータの設定
_8158_set_general_comparator	汎用コンパレータの設定
_8158_set_trigger_comparator	トリガ・コンパレータの設定
_8158_set_latch_source	カウンタのラッチ・タイミングの設定
_8158_set_ltc_logic	LTC 信号のロジックを設定
_8158_get_latch_data	ラッチ・データの取得

連続モーション、セクション 6.16

関数名	説明
_8158_set_continuous_move	絶対モーションの連続モーションを有効にする
_8158_check_continuous_buffer	バッファがエンプティかどうかを確認
_8158_dwell_move	タイマ移動の設定

多軸同時動作、セクション 6.17

関数名	説明
_8158_set_tr_move_all	多軸同時動作のセットアップ
_8158_set_ta_move_all	多軸同時動作のセットアップ
_8158_set_sr_move_all	多軸同時動作のセットアップ
_8158_set_sa_move_all	多軸同時動作のセットアップ
_8158_start_move_all	多軸台形プロファイル・モーションを開始
_8158_stop_move_all	多軸モーションの同時停止

汎用入力 / 出力、セクション 6.18

関数名	説明
_8158_set_gpio_output	デジタル出力の設定
_8158_get_gpio_output	デジタル出力を取得
_8158_get_gpio_input	デジタル入力を取得
_8158_set_gpio_input_function	信号タイプをデジタル入力に設定

ソフト・リミット、6.19

関数名	説明
_8158_disable_soft_limit	ソフト・リミット機能を使用しない
_8158_enable_soft_limit	ソフト・リミット機能を使用する
_8158_set_soft_limit	ソフト・リミットの設定

バックラッシュ補正 / 振動抑制、6.20

関数名	説明
_8158_backlash_comp	バックラッシュ調整パルスを補正に設定
_8158_suppress_vibration	振動抑制を無効なパルス・カウントに設定
_8158_set_fa_speed	FA 速度を原点モードに設定

速度プロファイルの計算、6.21

関数名	説明
_8158_get_tr_move_profile	相対台形速度プロファイルを取得
_8158_get_ta_move_profile	絶対台形速度プロファイルを取得

関数名	説明
_8158_get_sr_move_profile	相対 S 曲線速度プロファイルを取得
_8158_get_sa_move_profile	絶対 S 曲線速度プロファイルを取得

6.2 C/C++ プログラミング・ライブラリ

このセクションではすべての関数を詳しく説明します。関数のプロトタイプと一部のコマンド・データ・タイプは pci_8158.h で宣言されます。アプリケーション・プログラムでは上記のデータ・タイプを使用してください。データ・タイプ名とその範囲は下表の通りです。

タイプ名	説明	範囲
U8	8-bit ASCII 文字	0 ～ 255
I16	16-bit 符号化整数	-32768 ～ 32767
U16	16-bit 非符号化整数	0 ～ 65535
I32	32-bit 符号化長整数	-2147483648 ～ 2147483647
U32	32-bit 非符号化長整数	0 ～ 4294967295
F32	32-bit 単精度浮動点	-3.402823E38 ～ 3.402823E38
F64	64-bit 倍精度浮動点	-1.797683134862315E308 ～ 1.797683134862315E309
ブーリアン	ブーリアン論理値	真、偽

PCI-8158 のソフトウェア・ドライバの関数はフルネームを使用して、関数の実際の意味を表示します。命名規則は以下の通りです :

「C」プログラミング環境では :

_{ ハードウェアモデル }_{ アクション名 } すなわち、**_8158_initial()** となります。

C ライブラリと VB ライブラリの違いを示すため、それぞれの関数名の最初に、**B_8158_initial()** のように、大文字の「B」が配置されます。

6.3　システムおよび初期化

@ 名称

_8158_initial・　カードの初期化

_8158_close・　カードの終了

_8158_get_version・　ハードウェアおよびソフトウェア・バージョンの確認

_8158_set_security_key・　セキュリティ・パスワードの設定

_8158_check_security_key・　セキュリティ・パスワードの確認

_8158_reset_security_key・　セキュリティ・パスワードをデフォルト値にリセット

_8158_config_from_file Config・　ファイルから PCI-8158 を設定

@ 説明

_8158_initial:

この関数は 8158 カードを初期化してハードウェア・リソースを割り当てるのに使用されます。すべての 8158 カードはアプリケーションで他の関数を呼び出す前に、この関数を使って初期化しなければなりません。「**Manual_ID**」のパラメータを設定することで、カードの ID が手動または自動で割り当てられるタイプを選択できます。

_8158_close:

この関数は 8158 カードを終了し、アプリケーションの最後に呼び出されることになっていカードのリソースを解放するのに使用されます。

_8158_get_version:

ファ―ムウェア、ドライバ、DLL のバージョン情報を呼び出すことができます。

_8158_set_security_key:

この関数はセキュリティ・コードを PCI カードに設定するのに使用されます。

_8158_check_security_key, _8158_reset_security_key も参照してください。

_8158_check_security_key:

この関数は関数「_8158_set_security_key」で設定されるセキュリティ・コードを検証するのに使用されます。

_8158_set_security_key, _8158_reset_security_key も参照してください。

_8158_reset_security_key:

この関数を使えば、PCI カードのセキュリティ・コードをデフォルト値にリセットできます。デフォルトのセキュリティ・コードは 0 です。

_8158_check_security_key, _8158_set_security_key も参照してください。

_8158_config_from_file:

この関数は指定したファイルに従って PCI-8158 の設定をロードするのに使用されます。MotionCreatorPro を使えば、8158 のテストと設定を正しく行えます。設定の保存後、ファイルはユーザーのシステム・ディレクトリに 8158.ini として保存されます。

この関数を実行すると、システムのすべての 8158 カードは 8158.ini に記録されたパラメータに従って以下の関数を呼び出したように設定されます。

_8158_set_limit_logic
_8158_set_pcs_logic
_8158_set_ltc_logic
_8158_set_inp
_8158_set_erc
_8158_set_alm
_8158_set_pls_iptmode
_8158_set_pls_outmode
_8158_set_move_ratio
_8158_set_latch_source
_8158_set_feedback_src
_8158_set_home_config
_8158_set_soft_limit
_8158_set_fa_speed
_8158_set_sd

@ 文字列

C/C++(Windows 2000/XP)

I16 _8158_initial(I16 *CardID_InBit, I16 Manual_ID);
I16 _8158_close(void);
I16 _8158_get_version(I16 card_id, I16 *firmware_ver, I32 *driver_ver, I32 *dll_ver);
I16 _8158_set_security_key(I16 card_id, I16 old_secu_code, I16 new_secu_code);
I16 _8158_check_security_key(I16 card_id, I16 secu_code);
I16 _8158_reset_security_key(I16 card_id);
I16 _8158_config_from_file();

Visual Basic 6(Windows 2000/XP)

B_8158_initial(CardID_InBit As Integer, ByVal Manual_ID As Integer) As Integer
B_8158_close() As Integer
B_8158_get_version(ByVal card_id As Integer, firmware_ver As Integer, driver_ver As Long, dll_ver As Long) As Integer
B_8158_set_security_key(ByVal card_id As Integer, ByVal old_secu_code As Integer, ByVal new_secu_code As Integer) As Integer
B_8158_check_security_key(ByVal card_id As Integer, ByVal secu_code As Integer)As Integer
B_8158_reset_security_key(ByVal card_id As Integer);
B_8158_config_from_file() As Integer

@ 引数

CardID_InBit:

Manual_ID: オンボード DIP スイッチ（SW1）がカード ID を決定できるようにします。

値の意味 :

カード ID は以下で決定できます :

0: PCI スロットのシーケンス

1: オンボード DIP スイッチ（SW1）

card_id: PCI-8158 カードのインデックスを指定します。card_id は DIP スイッチ（SW1）によって決定されるか、スロットのシーケンスに依存します。_8158_initial() を参照してください。

firmware_ver: 現在のファームウェア・バージョン

driver_ver: 現在のデバイス・ドライバ・バージョン

dll_ver: 現在の DLL ライブラリのバージョン

old_secu_code: 以前のセキュリティ・コード

new_secu_code: 新しいセキュリティ・コード

secu_code: セキュリティ・コード

6.4　パルス入力 / 出力設定

@ 名称

_8158_set_pls_iptmode・　フィードバック・パルス入力を設定します。

_8158_set_pls_outmode・　パルス・コマンド出力を設定します。

_8158_set_feedback_src・　外部フィードバック・パルス入力を有効 / 無効にします。

@ 説明

_8158_set_pls_iptmode:

外部フィードバック・パルスの入力モードを設定します。フィードバック・パルス入力には 4 つのタイプがあります。この関数は、_8158_set_feedback_src() 関数の Src パラメータが使用可能な場合のみ有効です。

_8158_set_pls_outmode:

コマンド・パルスの出力モードを設定します。コマンド・パルス出力には 6 つのモードがあります。

_8158_set_feedback_src:

システムで外部エンコーダ・フィードバックが使用可能な場合、この関数の Src パラメータを Enabled（有効）状態に設定してください。すると、内部 28-bit アップ / ダウン・カウンタは _8158_set_pls_iptmode() 関数の設定に従ってカウントするようになります。また、カウンタはコマンド・パルス出力もカウントします。

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_set_pls_iptmode(I16 AxisNo, I16 pls_iptmode, I16 pls_logic);
I16 _8158_set_pls_outmode(I16 AxisNo, I16 pls_outmode);
I16 _8158_set_feedback_src(I16 AxisNo, I16 Src);
```

Visual Basic6 (Windows 2000/XP)

```
B_8158_set_pls_iptmode(ByVal AxisNo As Integer, ByVal pls_iptmode
        As Integer, ByVal pls_logic As Integer) As Integer
```

B_8158_set_pls_outmode(ByVal AxisNo As Integer, ByVal pls_outmode As Integer) As Integer
B_8158_set_feedback_src(ByVal AxisNo As Integer, ByVal Src As Integer) As Integer

@ 引数

AxisNo: パルス入力 / 出力を設定するように指定された軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

pls_iptmode: エンコーダのフィードバック・パルス入力モードの設定

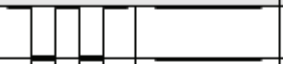
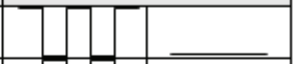
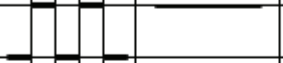
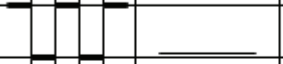
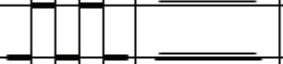
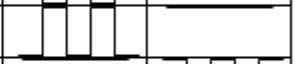
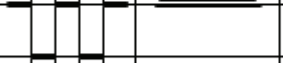
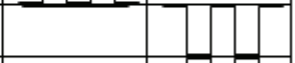
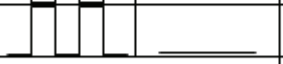
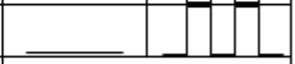
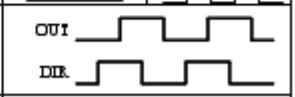
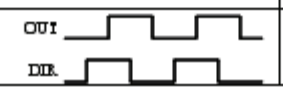
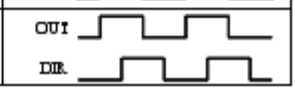
値	意味
0	1X A/B
1	2X A/B
2	4X A/B
3	CW/CCW

pls_logic: エンコーダのフィードバック・パルスのロジック

値	意味
0	逆方向ではない
1	逆方向

pls_outmode: コマンド・パルス出力モードの設定

Value meaning

value	type	Positive Direction	Negative Direction
0	OUT/DIR		
1	OUT/DIR		
2	OUT/DIR		
3	OUT/DIR		
4	CW / CCW		
5	CW / CCW		
6	AB		
7	AB		

Src: カウンタ・ソース

値	意味
0	外部フィードバック
1	コマンド・パルス

6.5　速度モード・モーション

@ 名称

_8158_tv_move・ 台形プロファイルで軸を一定の速度に加速

_8158_sv_move・ S 曲線プロファイルで軸を一定の速度に加速

_8158_emg_stop・ 直ちに停止

_8158_sd_stop・ 減速して、停止

_8158_get_current_speed・ 現在の速度を取得

_8158_speed_override・ オン・ザ・フライで速度を変更

_8158_set_max_override_speed・ 最大オーバーライド速度を設定

@ 説明

_8158_tv_move:

この関数は台形プロファイルで軸を一定の速度に加速します。軸は速度が変更されるか、軸が停止するように命令されるまで、一定の速度で移動します。方向は速度パラメータの符号によって決定されます。

_8158_sv_move:

この関数は S 曲線プロファイルで軸を一定の速度に加速します。軸は速度が変更されるか、軸が停止するように命令されるまで、一定の速度で移動します。方向は速度パラメータの符号によって決定されます。

_8158_emg_stop:

この関数は軸を直ちに停止させるのに使用されます。また、プリセットされた移動（台形および S 曲線の両方のモーション）、手動移動、または原点復帰機能を実行する場合にも役立ちます。

_8158_sd_stop:

この関数は台形またはS曲線プロファイルで軸を減速して停止させるのに使用されます。また、プリセットされた移動（台形および S 曲線の両方のモーション）、手動移動、または原点復帰機能を実行する場合にも役立ちます。注意：速度プロファイルは最初のモーション・プロファイルによって決定されます。

_8158_get_current_speed:

この機能は指定した軸の現在のパルス出力速度（パルス / 秒）を表示するのに使用されます。いつでもどんな動作モードでも使用できます。

_8158_speed_override:

（「_8158_tv_move」を実行している場合などの）モーション動作では、この関数はオン・ザ・フライで速度を変更するのに使用できます。セクション 4.2.14 を参照してください。

_8158_set_max_override_speed も参照してください。

8158_set_max_override_speed:

この関数は速度オーバーライド動作の前に最大オーバーライド速度（100% 速度）を設定するのに使用されます。セクション 4.2.14 を参照してください。

_8158_speed_override も参照してください。

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_tv_move(I16 AxisNo, F64 StrVel, F64 MaxVel, F64 Tacc);
I16 _8158_sv_move(I16 AxisNo, F64 StrVel, F64 MaxVel, F64 Tacc, F64
    SVacc);
I16 _8158_emg_stop(I16 AxisNo);
I16 _8158_sd_stop(I16 AxisNo, F64 Tdec);
I16 _8158_get_current_speed(I16 AxisNo, F64 *speed)
I16 _8158_set_max_override_speed(I16 AxisNo, F64 OvrdSpeed, I16
    Enable);
```

Visual Basic6 (Windows 2000/XP)

```
B_8158_tv_move(ByVal AxisNo As Integer, ByVal StrVel As Double,
    ByVal MaxVel As Double, ByVal Tacc As Double) As Integer
B_8158_sv_move(ByVal AxisNo As Integer, ByVal StrVel As Double,
    ByVal MaxVel As Double, ByVal Tacc As Double, ByVal SVacc
    As Double) As Integer
B_8158_emg_stop(ByVal AxisNo As Integer) As Integer
B_8158_sd_stop(ByVal AxisNo As Integer, ByVal Tdec As Double) As
    Integer
B_8158_get_current_speed(ByVal AxisNo As Integer, ByRef Speed As
    Double) As Integer
```

B_8158_set_max_override_speed(ByVal AxisNo As Integer, ByVal OvrdSpeed As Double, ByVal Enable As Integer) As Integer

@ 引数

AxisNo: 移動または停止するよう指定された軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

StrVel: 初速度（単位 : パルス / 秒）

MaxVel: 最大速度（単位 : パルス / 秒）

Tacc: 指定された加速時間（単位 : 秒）

SVacc: S 曲線加速が実行される指定された速度間隔

注意 : 純粋な S 曲線では SVacc = 0

Tdec: 指定された減速時間（単位 : 秒）

***Speed**: 現在の速度（パルス / 秒）を取得する変数

NewVelPercent: 最大オーバーライド速度（100% 速度）の割合

時間 : 速度をオーバライドする現在の速度の間隔単位 : 秒

OvrdSpeed: 最大オーバーライド速度（パルス / 秒）

Enabl: 0: 無効、1: オーバーライド速度動作を使用する

6.6　1 軸位置モード

@ 名称

_8158_start_tr_move・ 相対台形プロファイル移動を開始

_8158_start_ta_move・ 絶対台形プロファイル移動を開始

_8158_start_sr_move・ 相対 S 曲線プロファイル移動を開始

_8158_start_sa_move・ 絶対 S 曲線プロファイル移動を開始

_8158_set_move_ratio・ コマンド・パルスとフィードバック・パルスの比を設定

_8158_position_override・ オン・ザ・フライで位置を変更

@ 説明

共通:

移動方向はPosまたはDistパラメータの符号によって決定されます。指定速度に到達するのに移動距離が短すぎる場合、コントローラは MaxVel を自動的に引き下げます。また、Tacc、Tdec、VSacc、VSdec も短くなりますが、dV/dt（加速 / 減速）と d（dV/dt）/dt（加加速度）は変更されません。

_8158_start_tr_move:

この関数は軸を初速度（StrVel）から加速させ、一定速度（MaxVel）で回転させ、相対距離で停止するまで台形プロファイルで減速させます。加速（Tacc）および減速（Tdec）時間は独立して指定されます。また、プログラムはモーションの完了を待つ必要がなく、制御は直ちにプログラムに返されます。

_8158_start_ta_move:

この関数は軸を初速度（StrVel）から加速させ、一定速度（MaxVel）で回転させ、指定された絶対位置で停止するまで台形プロファイルで減速させます。加速（Tacc）および減速（Tdec）時間は独立して指定されます。また、プログラムはモーションの完了を待つ必要がなく、制御は直ちにプログラムに返されます。

_8158_start_sr_move:

この関数は軸を初速度（StrVel）から加速させ、一定速度（MaxVel）で回転させ、相対距離で停止するまで S 曲線プロファイルで減速させます。加速（Tacc）および減速（Tdec）時間は独立して指定されます。また、プログラムはモーションの完了を待つ必要がなく、制御は直ちにプログラムに返されます。

_8158_start_sa_move:

この関数は軸を初速度（StrVel）から加速させ、一定速度で回転させ、指定された絶対位置で停止するまで S 曲線プロファイルで減速させます。加速および減速時間は独立して指定されます。また、プログラムはモーションの完了を待つ必要がなく、制御は直ちにプログラムに返されます。

_8158_set_move_ratio:

この関数は指定した軸の定数を設定します。通常、軸の機械的解像度が異なる場合、軸は定数を必要とします。例えば、フィードバック・センサーの解像度がコマンド・プラスの解像度の 2 倍である場合、パラメータ「move_ratio」は 2 に設定されます。

_8158_position_override:

この関数はオン・ザ・フライでターゲット位置を変更するのに使用されます。この関数には幾つかの制限があります。使用する前にセクション 4.2.15 を参照してください。

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_start_tr_move(I16 AxisNo, F64 Dist, F64 StrVel, F64 MaxVel,
        F64 Tacc, F64 Tdec);
I16 _8158_start_ta_move(I16 AxisNo, F64 Pos, F64 StrVel, F64 MaxVel,
        F64 Tacc, F64 Tdec);
I16 _8158_start_sr_move(I16 AxisNo, F64 Dist, F64 StrVel, F64 MaxVel,
        F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_sa_move(I16 AxisNo, F64 Pos, F64 StrVel, F64 MaxVel,
        F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_set_move_ratio(I16 AxisNo, F64 move_ratio);
I16 _8158_position_override(I16 AxisNo, F64 NewPos);
```

Visual Basic6 (Windows 2000/XP)

B_8158_start_tr_move(ByVal AxisNo As Integer, ByVal Dist As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_ta_move(ByVal AxisNo As Integer, ByVal Pos As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_move(ByVal AxisNo As Integer, ByVal Dist As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8158_start_sa_move(ByVal AxisNo As Integer, ByVal Pos As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8158_set_move_ratio(ByVal AxisNo As Integer, ByVal move_ratio As Double) As Integer

B_8158_position_override(ByVal AxisNo As Integer, ByVal NewPos As Double) As Integer

@ 引数

AxisNo: 移動または位置を変更するよう指定された軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Dist: 指定された相対移動距離（単位 : パルス）

Pos: 指定された絶対移動位置（単位 : パルス）

StrVel: 速度プロファイルの初速度（単位 : パルス / 秒）

MaxVel: 最大速度（単位 : パルス / 秒）

Tacc: 指定された加速時間（単位 : 秒）

Tdec: 指定された減速時間（単位 : 秒）

SVacc: S 曲線加速が実行される指定された速度間隔

注意 : 純粋な S 曲線では SVacc = 0。詳しくはセクション 4.2.4 を参照

SVdec: S 曲線減速が実行される指定された速度間隔

注意 : 純粋な S 曲線では SVdec = 0。詳しくはセクション 4.2.4 を参照

Move_ratio: （フィードバックの解像度）/（コマンドの解像度）の比率、0 とはならない

NewPos: 指定された新しい絶対移動位置

6.7　直線補間モーション

@ 名称

_8158_start_tr_move_xy・ 台形プロファイルで X および Y 軸の相対 2 軸直線補間を開始

_8158_start_ta_move_xy・ 台形プロファイルで X および Y 軸の絶対 2 軸直線補間を開始

_8158_start_sr_move_xy・ S 曲線プロファイルで X および Y 軸の相対 2 軸直線補間を開始

_8158_start_sa_move_xy・ S 曲線プロファイルで X および Y 軸の絶対 2 軸直線補間を開始

_8158_start_tr_move_zu・ 台形プロファイルで Z および U 軸の相対 2 軸直線補間を開始

_8158_start_ta_move_zu・ 台形プロファイルで Z および U 軸の絶対 2 軸直線補間を開始

_8158_start_sr_move_zu・ S 曲線プロファイルで Z および U 軸の相対 2 軸直線補間を開始

_8158_start_sa_move_zu・ S 曲線プロファイルで Z および U 軸の絶対 2 軸直線補間を開始

_8158_start_tr_move_ab・ 台形プロファイルで A および B 軸の相対 2 軸直線補間を開始

_8158_start_ta_move_ab・ 台形プロファイルで A および B 軸の絶対 2 軸直線補間を開始

_8158_start_sr_move_ab・ S 曲線プロファイルで A および B 軸の相対 2 軸直線補間を開始

_8158_start_sa_move_ab・ S 曲線プロファイルで A および B 軸の絶対 2 軸直線補間を開始

_8158_start_tr_move_cd・ 台形プロファイルで C および D 軸の相対 2 軸直線補間を開始

_8158_start_ta_move_cd・ 台形プロファイルで C および D 軸の絶対 2 軸直線補間を開始

_8158_start_sr_move_cd・ S 曲線プロファイルで C および D 軸の相対 2 軸直線補間を開始

_8158_start_sa_move_cd・ S曲線プロファイルでCおよびD軸の絶対2軸直線補間を開始

_8158_start_tr_line2・ 4軸の任意の2軸に対して台形プロファイルで相対2軸直線補間を開始

_8158_start_ta_line2・ 4軸の任意の2軸に対して台形プロファイルで絶対2軸直線補間を開始

_8158_start_sr_line2・ 4軸の任意の2軸に対してS曲線プロファイルで相対2軸直線補間を開始

_8158_start_sa_line2・ 4軸の任意の2軸に対してS曲線プロファイルで絶対2軸直線補間を開始

_8158_start_tr_line3・ 4軸の任意の3軸に対して台形プロファイルで相対3軸直線補間を開始

_8158_start_ta_line3・ 4軸の任意の3軸に対して台形プロファイルで相対3軸直線補間を開始

_8158_start_sr_line3・ 4軸の任意の3軸に対してS曲線プロファイルで相対3軸直線補間を開始

_8158_start_sa_line3・ 4軸の任意の3軸に対してS曲線プロファイルで絶対3軸直線補間を開始

_8158_start_tr_line4・ 4軸の任意の4軸に対して台形プロファイルで相対4軸直線補間を開始

_8158_start_ta_line4・ 4軸の任意の4軸に対して台形プロファイルで絶対4軸直線補間を開始

_8158_start_sr_line4・ 4軸の任意の4軸に対してS曲線プロファイルで相対4軸直線補間を開始

_8158_start_sa_line4・ 4軸の任意の4軸に対してS曲線プロファイルで絶対4軸直線補間を開始

@ 説明

上記関数は様々なプロファイルで直線補間モーションを実行します。関数の詳しい比較表は以下の通りです。

関数	軸総数	速度プロファイル	相対 / 絶対	ターゲット軸
_8158_start_tr_move_xy	2	T	R	軸 0 および 1
_8158_start_ta_move_xy	2	T	A	軸 0 および 1
_8158_start_sr_move_xy	2	S	R	軸 0 および 1
_8158_start_sa_move_xy	2	S	A	軸 0 および 1
_8158_start_tr_move_zu	2	T	R	軸 2 および 3
_8158_start_ta_move_zu	2	T	A	軸 2 および 3
_8158_start_sr_move_zu	2	S	R	軸 2 および 3
_8158_start_sa_move_zu	2	S	A	軸 2 および 3
_8158_start_tr_move_ab	2	T	R	軸 4 および 5
_8158_start_ta_move_ab	2	T	A	軸 4 および 5
_8158_start_sr_move_ab	2	S	R	軸 4 および 5
_8158_start_sa_move_ab	2	S	A	軸 4 および 5
_8158_start_tr_move_cd	2	T	R	軸 6 および 7
_8158_start_ta_move_cd	2	T	A	軸 6 および 7
_8158_start_sr_move_cd	2	S	R	軸 6 および 7
_8158_start_sa_move_cd	2	S	A	軸 6 および 7

関数	軸総数	速度プロファイル	相対 / 絶対	ターゲット軸
_8158_start_tr_line2	2	T	R	4 軸の任意の 2 軸
_8158_start_ta_line2	2	T	A	4 軸の任意の 2 軸
_8158_start_sr_line2	2	S	R	4 軸の任意の 2 軸
_8158_start_sa_line2	2	S	A	4 軸の任意の 2 軸

íçà”: 直線補間のターゲットの 2 軸はカードの前 0-3 軸のうちの 2 軸か後 4-7 軸のうちの 2 軸です。それら 2 つのグループをまたぐことはできません。

関数	軸総数	速度プロファイル	相対 / 絶対	ターゲット軸
_8158_start_tr_line3	3	T	R	4 軸の任意の 3 軸
_8158_start_ta_line3	3	T	A	4 軸の任意の 3 軸
_8158_start_sr_line3	3	S	R	4 軸の任意の 3 軸

関数	軸総数	速度プロファイル	相対 / 絶対	ターゲット軸
_8158_start_sa_line3	3	S	A	4 軸の任意の 3 軸

íçà": 直線補間のターゲットの 3 軸はカードの前 4 軸のうちの 3 軸か後 4 軸のうちの 3 軸です。それら 2 つのグループをまたぐことはできません。

関数	軸総数	速度プロファイル	相対 / 絶対	ターゲット軸
_8158_start_tr_line4	4	T	R	4 軸の任意の 4 軸
_8158_start_ta_line4	4	T	A	4 軸の任意の 4 軸
_8158_start_sr_line4	4	S	R	4 軸の任意の 4 軸
_8158_start_sa_line4	4	S	A	4 軸の任意の 4 軸

íçà": 直線補間のターゲットの 4 軸はカードの前 4 軸のうちの 4 軸か後 4 軸のうちの 4 軸です。それら 2 つのグループをまたぐことはできません。

速度プロファイル :

T: 台形プロファイル

S: S 曲線プロファイル

相対 / 絶対

R: 相対距離

A: 絶対位置

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_start_tr_move_xy(I16 Card_id, F64 DistX, F64 DistY, F64
        StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8158_start_ta_move_xy(I16 Card_id, F64 PosX, F64 PosY, F64
        StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
```

I16 _8158_start_sr_move_xy(I16 Card_id, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8158_start_sa_move_xy(I16 Card_id, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8158_start_tr_move_zu(I16 Card_id, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8158_start_ta_move_zu(I16 Card_id, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8158_start_sr_move_zu(I16 Card_id, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8158_start_sa_move_zu(I16 Card_id, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8158_start_tr_move_ab(I16 Card_id, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8158_start_ta_move_ab(I16 Card_id, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8158_start_sr_move_ab(I16 Card_id, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8158_start_sa_move_ab(I16 Card_id, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8158_start_tr_move_cd(I16 Card_id, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8158_start_ta_move_cd(I16 Card_id, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8158_start_sr_move_cd(I16 Card_id, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8158_start_sa_move_cd(I16 Card_id, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8158_start_tr_line2(I16 *AxisArray, F64 *DistArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8158_start_ta_line2(I16 *AxisArray, F64 *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8158_start_sr_line2(I16 *AxisArray, F64 *DistArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8158_start_sa_line2(I16 *AxisArray, F64 *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8158_start_tr_line3(I16 *AxisArray, F64 *DistArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8158_start_ta_line3(I16 *AxisArray, F64 *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8158_start_sr_line3(I16 *AxisArray, F64 *DistArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_sa_line3(I16 *AxisArray, F64 *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_tr_line4(I16 *AxisArray, F64 *DistArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8158_start_ta_line4(I16 *AxisArray, F64 *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8158_start_sr_line4(I16 *AxisArray, F64 *DistArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8158_start_sa_line4(I16 *AxisArray, F64 *PosArray, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

Visual Basic6 (Windows 2000/XP)

B_8158_start_tr_move_xy(ByVal Card_id As Integer, ByVal DistX As Double, ByVal DistY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer
B_8158_start_ta_move_xy(ByVal Card_id As Integer, ByVal PosX As Double, ByVal PosY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer
B_8158_start_sr_move_xy(ByVal Card_id As Integer, ByVal DistX As Double, ByVal DistY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer
B_8158_start_sa_move_xy(ByVal Card_id As Integer, ByVal PosX As Double, ByVal PosY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer
B_8158_start_tr_move_zu(ByVal Card_id As Integer, ByVal DistX As Double, ByVal DistY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double);
B_8158_start_ta_move_zu(ByVal Card_id As Integer, ByVal PosX As Double, ByVal PosY As Double, ByVal StrVel As Double, ByVal

MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_move_zu(ByVal Card_id As Integer, ByVal DistX As Double, ByVal DistY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8158_start_sa_move_zu(ByVal Card_id As Integer, ByVal PosX As Double, ByVal PosY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8158_start_tr_move_ab(ByVal Card_id As Integer, ByVal DistX As Double, ByVal DistY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double);

B_8158_start_ta_move_ab(ByVal Card_id As Integer, ByVal PosX As Double, ByVal PosY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_move_ab(ByVal Card_id As Integer, ByVal DistX As Double, ByVal DistY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8158_start_sa_move_ab(ByVal Card_id As Integer, ByVal PosX As Double, ByVal PosY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8158_start_tr_move_cd(ByVal Card_id As Integer, ByVal DistX As Double, ByVal DistY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double);

B_8158_start_ta_move_cd(ByVal Card_id As Integer, ByVal PosX As Double, ByVal PosY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_move_cd(ByVal Card_id As Integer, ByVal DistX As Double, ByVal DistY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8158_start_sa_move_cd(ByVal Card_id As Integer, ByVal PosX As Double, ByVal PosY As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8158_start_tr_line2(AxisArray() As Integer, DistArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_ta_line2(AxisArray() As Integer, PosArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_line2((AxisArray() As Integer, DistArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_sa_line2(AxisArray() As Integer, PosArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_tr_line3(AxisArray() As Integer, DistArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_ta_line3(AxisArray() As Integer, PosArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_line3((AxisArray() As Integer, DistArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_sa_line3(AxisArray() As Integer, PosArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_tr_line4(AxisArray() As Integer, DistArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_ta_line4(AxisArray() As Integer, PosArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_line4((AxisArray() As Integer, DistArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_sa_line4(AxisArray() As Integer, PosArray() As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

@ 引数

card_id: PCI-8158 カードのインデックスを指定します。card_id は DIP スイッチ（SW1）によって決定されるか、スロットのシーケンスに依存します。_8158_initial() を参照してください。

AxisNo: 移動または位置を変更するよう指定された軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

DistX: 軸 0 の指定された相対移動距離（単位 : パルス）

DistY: 軸 1 の指定された相対移動距離（単位 : パルス）

PosX: 軸 0 の指定された絶対移動位置（単位 : パルス）

PosY: 軸 1 の指定された絶対移動位置（単位 : パルス）

StrVel: 速度プロファイルの初速度（単位 : パルス / 秒）

MaxVel: 最大速度（単位 : パルス / 秒）

Tacc: 指定された加速時間（単位 : 秒）

Tdec: 指定された減速時間（単位 : 秒）

SVacc: S 曲線加速が実行される指定された速度間隔

注意 : 純粋な S 曲線では SVacc = 0。詳しくはセクション 4.2.4 を参照

SVdec: S 曲線減速が実行される指定された速度間隔

注意 : 純粋な S 曲線では SVdec = 0。詳しくはセクション 4.2.4 を参照

***AxisArray**: 補間を実行する軸番号の配列

例 : I16 AxisArray[2] = {0, 3}; // 軸 0 および軸 3（正）

I16 AxisArray[3] = {0,2, 3}; // 軸 0、2、3（正）

I16 AxisArray[2] = {1, 6}; // 軸 1 および軸 6（誤）

***DistArray**: 直線補間の相対距離の配列

例 : I16 AxisArray[2] = {0, 3}; // 軸 0 および軸 3

F64 DistArray[2] = {1000.0, 2000.0} // 軸 0 および 3 の場合

***PosArray**: 直線補間の絶対位置の配列

例 : I16 AxisArray[3] = {0.2, 3}; // 軸 0、2、3

F64 PosArray[3] = {200.0, 300.0, 400.0} // 軸 0、2、3 の絶対位置

6.8　円形補間モーション

@ 名称

_8158_start_tr_arc_xy・ Ｘおよび Ｙ軸の Ｔ 曲線相対円形補間を開始

_8158_start_ta_arc_xy・ Ｘおよび Ｙ軸の Ｔ 曲線絶対円形補間を開始

_8158_start_sr_arc_xy・ Ｘ および Ｙ 軸の Ｓ 曲線相対円形補間を開始

_8158_start_sa_arc_xy・ Ｘ および Ｙ 軸の Ｓ 曲線絶対円形補間を開始

_8158_start_tr_arc_zu・ Ｚおよび Ｕ軸の Ｔ 曲線相対円形補間を開始

_8158_start_ta_arc_zu・ Ｚおよび Ｕ軸の Ｔ 曲線絶対円形補間を開始

_8158_start_sr_arc_zu・ Ｚ および Ｕ 軸の Ｓ 曲線相対円形補間を開始

_8158_start_sa_arc_zu・ Ｚ および Ｕ 軸の Ｓ 曲線絶対円形補間を開始

_8158_start_tr_arc_ab・ Ａおよび Ｂ軸の Ｔ 曲線相対円形補間を開始

_8158_start_ta_arc_ab・ Ａおよび Ｂ軸の Ｔ 曲線絶対円形補間を開始

_8158_start_sr_arc_b・ Ａ および Ｂ 軸の Ｓ 曲線相対円形補間を開始

_8158_start_sa_arc_ab・ Ａ および Ｂ 軸の Ｓ 曲線絶対円形補間を開始

_8158_start_tr_arc_cd・ Ｃおよび Ｄ軸の Ｔ 曲線相対円形補間を開始

_8158_start_ta_arc_cd・ Ｃおよび Ｄ軸の Ｔ 曲線絶対円形補間を開始

_8158_start_sr_arc_cd・ Ｃ および Ｄ 軸の Ｓ 曲線相対円形補間を開始

_8158_start_sa_arc_cd・ Ｃ および Ｄ 軸の Ｓ 曲線絶対円形補間を開始

_8158_start_tr_arc2・ ４軸の任意の 2 軸に対して Ｔ 曲線相対円形補間を開始

_8158_start_ta_arc2・ ４ 軸の任意の 2 軸に対して Ｔ 曲線絶対円形補間を開始

_8158_start_sr_arc2・ ４軸の任意の 2 軸に対して Ｓ 曲線相対円形補間を開始

_8158_start_sa_arc2・ ４軸の任意の 2 軸に対して Ｓ 曲線絶対円形補間を開始

@ 説明

上記関数は様々なプロファイルで円形補間モーションを実行します。関数の詳しい比較表は以下の通りです。

関数	軸総数	速度プロファイル	相対 / 絶対	ターゲット軸
_8158_start_tr_arc_xy	2	台形	R	軸 0 および 1
_8158_start_ta_arc_xy	2	台形	A	軸 0 および 1
_8158_start_sr_arc_xy	2	S 曲線	R	軸 0 および 1
_8158_start_sa_arc_xy	2	S 曲線	A	軸 0 および 1
_8158_start_tr_arc_zu	2	台形	R	軸 2 および 3
_8158_start_ta_arc_zu	2	台形	A	軸 2 および 3
_8158_start_sr_arc_zu	2	S 曲線	R	軸 2 および 3
_8158_start_sa_arc_zu	2	S 曲線	A	軸 2 および 3
_8158_start_tr_arc_ab	2	台形	R	軸 4 および 5
_8158_start_ta_arc_ab	2	台形	A	軸 4 および 5
_8158_start_sr_arc_ab	2	S 曲線	R	軸 4 および 5
_8158_start_sa_arc_ab	2	S 曲線	A	軸 4 および 5
_8158_start_tr_arc_cd	2	台形	R	軸 6 および 7
_8158_start_ta_arc_cd	2	台形	A	軸 6 および 7
_8158_start_sr_arc_cd	2	S 曲線	R	軸 6 および 7
_8158_start_sa_arc_cd	2	S 曲線	A	軸 6 および 7

関数	軸総数	速度プロファイル	相対 / 絶対	ターゲット軸
_8158_start_tr_arc2	2	台形	R	4 軸の任意の 2 軸
_8158_start_ta_arc2	2	台形	A	4 軸の任意の 2 軸
_8158_start_sr_arc2	2	S 曲線	R	4 軸の任意の 2 軸
_8158_start_sa_arc2	2	S 曲線	A	4 軸の任意の 2 軸

íçà”: 直線補間のターゲットの 2 軸はカードの前 4 軸（0-3）のうちの 2 軸か後 4 軸（4-7）のうちの 2 軸です。それら 2 つのグループをまたぐことはできません。

@ 文字列

C/C++(Windows 2000/XP)

I16 _8158_start_tr_arc_xy(I16 card_id, F64 OffsetCx, F64 OffsetCy, F64 OffsetEx, F64 OffsetEy, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);

I16 _8158_start_ta_arc_xy(I16 card_id, F64 Cx, F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);

I16 _8158_start_sr_arc_xy(I16 card_id, F64 OffsetCx, F64 OffsetCy, F64 OffsetEx, F64 OffsetEy, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64 SVdec);

I16 _8158_start_sa_arc_xy(I16 card_id, F64 Cx, F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64 SVdec);

I16 _8158_start_tr_arc_zu(I16 card_id, F64 OffsetCx, F64 OffsetCy, F64 OffsetEx, F64 OffsetEy, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);

I16 _8158_start_ta_arc_zu(I16 card_id, F64 Cx, F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);

I16 _8158_start_sr_arc_zu(I16 card_id, F64 OffsetCx, F64 OffsetCy, F64 OffsetEx, F64 OffsetEy, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64 SVdec);

I16 _8158_start_sa_arc_zu(I16 card_id, F64 Cx, F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64 SVdec);

I16 _8158_start_tr_arc_ab(I16 card_id, F64 OffsetCx, F64 OffsetCy, F64 OffsetEx, F64 OffsetEy, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);

I16 _8158_start_ta_arc_ab(I16 card_id, F64 Cx, F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);

I16 _8158_start_sr_arc_ab(I16 card_id, F64 OffsetCx, F64 OffsetCy, F64 OffsetEx, F64 OffsetEy, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64 SVdec);

I16 _8158_start_sa_arc_ab(I16 card_id, F64 Cx, F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64 SVdec);

I16 _8158_start_tr_arc_cd(I16 card_id, F64 OffsetCx, F64 OffsetCy, F64 OffsetEx, F64 OffsetEy, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);

I16 _8158_start_ta_arc_cd(I16 card_id, F64 Cx, F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);

I16 _8158_start_sr_arc_cd(I16 card_id, F64 OffsetCx, F64 OffsetCy, F64 OffsetEx, F64 OffsetEy, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64 SVdec);

I16 _8158_start_sa_arc_cd(I16 card_id, F64 Cx, F64 Cy, F64 Ex, F64 Ey, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec,F64 SVacc,F64 SVdec);

I16 _8158_start_tr_arc2(I16 *AxisArray, F64 *OffsetCenter, F64 *OffsetEnd, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);

I16 _8158_start_ta_arc2(I16 *AxisArray, F64 *CenterPos, F64 *EndPos, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec);

I16 _8158_start_sr_arc2(I16 *AxisArray, F64 *OffsetCenter, F64 *OffsetEnd, I16 CW_CCW, F64 StrVel,F64 MaxVel,F64 Tacc,F64 Tdec, F64 SVacc,F64 SVdec);

I16 _8158_start_sa_arc2(I16 *AxisArray, F64 *CenterPos, F64 *EndPos, I16 CW_CCW, F64 StrVel,F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

Visual Basic6 (Windows 2000/XP)

B_8158_start_tr_arc_xy(ByVal card_id As Integer, ByVal OffsetCx As Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double, ByVal OffsetEy As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double);

B_8158_start_ta_arc_xy(ByVal card_id As Integer, ByVal Cx As Double, ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_arc_xy(ByVal card_id As Integer, ByVal OffsetCx As Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double, ByVal OffsetEy As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_sa_arc_xy(ByVal card_id As Integer, ByVal Cx As Double, ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_tr_arc_zu(ByVal card_id As Integer, ByVal OffsetCx As Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double,

ByVal OffsetEy As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double);

B_8158_start_ta_arc_zu(ByVal card_id As Integer, ByVal Cx As Double, ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_arc_zu(ByVal card_id As Integer, ByVal OffsetCx As Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double, ByVal OffsetEy As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_sa_arc_zu(ByVal card_id As Integer, ByVal Cx As Double, ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_tr_arc_ab(ByVal card_id As Integer, ByVal OffsetCx As Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double, ByVal OffsetEy As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double);

B_8158_start_ta_arc_ab(ByVal card_id As Integer, ByVal Cx As Double, ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_arc_ab(ByVal card_id As Integer, ByVal OffsetCx As Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double, ByVal OffsetEy As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_sa_arc_ab(ByVal card_id As Integer, ByVal Cx As Double, ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_tr_arc_cd(ByVal card_id As Integer, ByVal OffsetCx As Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double,

ByVal OffsetEy As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double);

B_8158_start_ta_arc_cd(ByVal card_id As Integer, ByVal Cx As Double, ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_arc_cd(ByVal card_id As Integer, ByVal OffsetCx As Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double, ByVal OffsetEy As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_sa_arc_cd(ByVal card_id As Integer, ByVal Cx As Double, ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double, ByVal CW_CCW As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_tr_arc2(AxisArray() As Integer, OffsetCenter() As Double, OffsetEnd() As Double, Byval CW_CCW As Integer, ByVal StrVel As Double , ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_ta_arc2(AxisArray() As Integer, CenterPos() As Double, EndPos() As Double, Byval CW_CCW As Integer, ByVal StrVel As Double , ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8158_start_sr_arc2(AxisArray() As Integer, OffsetCenter() As Double, OffsetEnd() As Double, Byval CW_CCW As Integer, ByVal StrVel As Double , ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

B_8158_start_sa_arc2(AxisArray() As Integer, CenterPos() As Double, EndPos() As Double, Byval CW_CCW As Integer, ByVal StrVel As Double , ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal Svacc As Double, ByVal Svdec As Double) As Integer

@ Argument

card_id: Specify the PCI-8158 card index. The card_id could be decided by DIP switch (SW1) or depend on slot sequence. Please refer to _8158_initial().

AxisNo: Axis number designated to move or change position.

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

OffsetCx: 中心への X 軸（ターゲット軸の最初の軸）のオフセット

OffsetCy: 中心への Y 軸(ターゲット軸の 2 番目の軸)のオフセット

OffsetEx: 円弧終点への X 軸（ターゲット軸の最初の軸）のオフセット

OffsetEy: 円弧終点への Y 軸のオフセット

Cx: 円弧中心の X 軸（ターゲット軸の最初の軸）の絶対位置

Cy: 円弧中心の Y 軸（ターゲット軸の 2 番目の軸）の絶対位置

Ex: 円弧終点の X 軸（ターゲット軸の最初の軸）の絶対位置

Ey: 円弧終点の Y 軸（ターゲット軸の 2 番目の軸）の絶対位置

CW_CCW: 円弧の指定方向

値	意味
0	時計回り（CW）
1	反時計回り（CCW）

StrVel: 速度プロファイルの初速度（単位 : パルス / 秒）

MaxVel: 最大速度（単位 : パルス / 秒）

Tacc: 指定された加速時間（単位 : 秒）

Tdec: 指定された減速時間（単位 : 秒）

SVacc: S 曲線加速が実行される指定された速度間隔

注意 : 純粋な S 曲線では SVacc = 0。詳しくはセクション 4.2.4 を参照

SVdec: S 曲線減速が実行される指定された速度間隔

注意 : 純粋な S 曲線では SVdec = 0。詳しくはセクション 4.2.4 を参照

***AxisArray**: 補間を実行する軸番号の配列

例 : I16 AxisArray[2] = {0, 3}; // 軸 0 および軸 3（正）

I16 AxisArray[2] = {1, 6}; // 軸 1 および軸 6（誤）

***OffsetCenter**: （開始位置に対する）中心へのオフセットの配列

例 : F64 OffsetCenter[2] = {2000.0, 0.0}; // 最初の軸と 2 番目の軸に対する開始位置（始点）からのオフセット

***OffsetEnd**: （開始位置に対する）円弧終点へのオフセットの配列

例 : F64 OffsetEnd[2] = {4000.0, 0.0}; // 最初の軸と 2 番目の軸に対する開始位置（始点）からのオフセット

***CenterPos**: 円弧絶対位置の中心の配列

例 : F64 CenterPos[2] = {2000.0, 0.0}; // 最初の軸と 2 番目の軸に対する絶対中心位置

***EndPos**: 円弧絶対位置の終点の配列

例 : F64 EndPos[2] = {4000.0, 0.0}; // 最初の軸と 2 番目の軸に対する絶対終点位置

6.9 原点復帰モード

@ 名称

_8158_set_home_config・ 原点復帰移動モーションを設定

_8158_home_move・ 原点復帰移動を実行

_8158_home_search・ 原点自動検索を実行

@ 説明

_8158_set_home_config:

home_move() 関数の原点復帰モード、原点（ORG）およびインデックス信号（EZ）のロジック、EZ カウント、ERC 出力オプションを設定します。home_mode 制御の設定についてはセクション 4.2.10 を参照してください。

_8158_home_move:

この関数は _8164_set_home_config() 関数の設定に従って軸に原点復帰移動を実行させます。移動の方向は速度パラメータ（MaxVel）の符号によって決定されます。この関数の停止条件は home_mode の設定によって決まるので、最初の移動方向を選択には注意が必要です。また、リミット・スイッチに接触するときの条件や軸を停止させることが可能な他の条件の処理にも注意が必要です。詳しくはセクション 4.2.10 を参照してください。

_8158_home_search:

この関数は _8164_set_home_config() 関数の設定に従って軸に原点検索移動を実行させます。移動の方向は速度パラメータ（MaxVel）の符号によって決定されます。この関数の停止条件は home_mode の設定によって決まるので、最初の移動方向を選択には注意が必要です。また、リミット・スイッチに接触するときの条件や軸を停止させることが可能な他の条件の処理にも注意が必要です。詳しくはセクション 4.2.11 を参照してください。

@ 文字列

C/C++(Windows 2000/XP)

I16 _8158_set_home_config(I16 AxisNo, I16 home_mode, I16 org_logic, I16 ez_logic, I16 ez_count, I16 erc_out);
I16 _8158_home_move(I16 AxisNo, F64 StrVel, F64 MaxVel, F64 Tacc);
I16 _8158_home_search(I16 AxisNo, F64 StrVel, F64 MaxVel, F64 Tacc, F64 ORGOffset);

Visual Basic (Windows 2000/XP)

B_8158_set_home_config(ByVal AxisNo As Integer, ByVal home_mode As Integer, ByVal org_logic As Integer, ByVal ez_logic As Integer, ByVal ez_count As Integer, ByVal erc_out As Integer) As Integer
B_8158_home_move(ByVal AxisNo As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double) As Integer
B_8158_home_search(ByVal AxisNo As Integer, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal ORGOffset As Double) As Integer

@ 引数

AxisNo: 移動または位置を変更するよう指定された軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

home_mode: 原点復帰の停止モード。値は 0 から 12 までです。セクション 4.2.10 をご覧ください。

org_logic: ORG のアクション・ロジックの設定

値	意味
0	アクティブ低

値	意味
1	アクティブ高

ez_logic: EZ のアクション・ロジックの設定

値	意味
0	アクティブ低
1	アクティブ高

ez_count: 0–15（セクション 4.2.10 参照）

erc_out: ERC の出力オプションを設定

値	意味
0	非 ERC 出力
1	原点移動完了時 ERC 信号を出力

StrVel: 速度プロファイルの初速度（単位 : パルス / 秒）

MaxVel: 最大速度（単位 : パルス / 秒）

Tacc: 指定された加速時間（単位 : 秒）

ORGOffset: 原点検索が ORG 信号に達した時のエスケープ・パルス量（単位 : パルス）

6.10 手動パルス・モーション

@ 名称

_8158_disable_pulser_input・ パルス入力を無効にする

_8158_pulser_pmove・ 手動パルス p_move

_8158_pulser_vmove・ 手動パルス v_move

_8158_set_pulser_ratio・ 手動パルス比を実際の出力パルス速度に設定

_8158_set_pulser_iptmode・ パルスの入力信号モードを設定

@ 説明

_8158_disable_pulser_input

この関数はパルス出力を有効または無効にするのに使用されます。

_8158_pulser_pmove

このコマンドで、軸は手動パルス入力に従って移動を開始します。_8158_disable_pulser_input 関数がパルスを無効にするか、出力パルス数が距離に到達するまで、軸は 1 つの手動パルスを受信すると 1 つのパルスを出力します。

_8158_pulser_vmove

このコマンドで、軸は手動パルス入力に従って移動を開始します。_8158_disable_pulser_input 関数がパルスを無効にするまで、軸は1つの手動パルスを受信すると1つのパルスを出力します。

_8158_set_pulser_ratio

手動パルス比を実際の出力パルス速度に設定します。手動パルス出力速度の方程式や以下の通りです :

出力パルス・カウント = 入力パルス・カウント × 4 （MultiF +1） × （DivF +1）/2048

DivF = 0 ～ 2047 除算係数

MultiF= 0 ～ 31 乗算計数

_8158_set_pulser_iptmode

この館数は手動パルスの入力モードを設定するのに使用されます。

@ 文字列

C/C++(Windows 2000/XP)

I16 _8158_disable_pulser_input(I16 AxisNo, U16 Disable);
I16 _8158_pulser_pmove(I16 AxisNo, F64 Dist, F64 SpeedLimit);
I16 _8158_pulser_vmove(I16 AxisNo, F64 SpeedLimit);
I16 _8158_set_pulser_ratio(I16 AxisNo, I16 DivF, I16 MultiF);
I16 _8158_set_pulser_iptmode(I16 AxisNo, I16 InputMode, I16 **Inverse);**

Visual Basic (Windows 2000/XP)

B_8158_disable_pulser_input(ByVal AxisNo As Integer, ByVal Disable As Integer) As Integer
B_8158_pulser_pmove(ByVal AxisNo As Integer, ByVal Dist As Double, ByVal SpeedLimit As Double) As Integer
B_8158_pulser_vmove(ByVal AxisNo As Integer, ByVal SpeedLimit As Double) As Integer
B_8158_set_pulser_ratio(ByVal AxisNo As Integer, ByVal DivF As Integer, ByVal MultiF As Integer) As Integer
B_8158_set_pulser_iptmode(ByVal AxisNo As Integer, ByVal InputMode As Integer, ByVal Inverse As Integer) As Integer

@ 引数

AxisNo: 移動または位置を変更するよう指定された軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Disable: パルス入力を無効にする

Disable = 1、パルスを使用しない

Disable = 0、パルスを使用する

Dist: 指定された相対移動距離（単位 : パルス）

例えば、SpeedLimit を 100pps に設定すると、入力パルス信号速度が 100pps を上回っても、軸は最速の 100pps を超えて移動することはできません。

DivF: 除算係数（0-2047）

MultiF: 乗算係数（0-31）

InputMode: PA ピンと PB ピンからの手動パルス入力モードの設定

値	意味
0	1X AB 相タイプのパルス入力
1	2X AB 相タイプのパルス入力
2	4X AB 相タイプのパルス入力
3	CW/CCW タイプのパルス入力

Inverse: 移動方向をパルス方向と逆にする

値	意味
0	逆にしない
1	移動方向を逆にする

6.11 モーション・ステータス

@ 名称

_8102_motion_done – モーション・ステータスを返す

@ 説明

_8102_motion_done:

8102 のモーション・ステータスを返します。応答コードは以下の通りです :

0	通常の停止状態
1	DR を待機
2	CSTA 入力を待機
3	内部同期信号を待機
4	他の軸が停止するのを待機
5	ERC タイマの完了を待機
6	方向変更タイマの完了を待機
7	バックラックの訂正中
8	PA/PB を待機
9	FA 速度で動作
10	FL 速度で動作
11	加速中
12	FH 速度で動作
13	減速中
14	INP を待機
15	その他（制御開始）
16	SALM
17	SPEL
18	SMEL
19	SEMG
20	SSTP
21	SERC

@ 文字列

C/C++(Windows 2000/XP)

I16 _8102_motion_done(I16 AxisNo)

Visual Basic (Windows 2000/XP)

B_8102_motion_done(ByVal AxisNo As Integer) As Integer

@ 引数

AxisNo: 移動または位置を変更するよう指定された軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

6.12 モーション・インタフェース I/O

@ 名称

_8158_set_servo・ SVON 信号のオン－オフ状態を設定

_8158_set_pcs_logic・ PCS 信号のロジックを設定

_8158_set_pcs・ PCS の位置オーバーライドを可能にする

_8158_set_clr_mode・ CLR 信号のモードを設定

_8158_set_inp・ INP 信号および動作モードのロジックを設定

_8158_set_alm・ ALM 信号および動作モードのロジックを設定

_8158_set_erc・ ERC 信号および動作モードのロジックを設定

_8158_set_erc_out・ ERC 信号を出力

_8158_clr_erc・ ERC 信号をクリア

_8158_set_sd・ SD 信号および動作モードのロジックを設定

_8158_enable_sd・ SD 信号を使用する

_8158_set_limit_logic・ PEL/MEL 信号のロジックを設定

_8158_set_limit_mode・ PEL/MEL の動作モードを設定

_8158_get_io_status・ 各 8158 のすべてのモーション I/O ステータスを取得

@ 説明

_8158_set_servo:

この関数で SVON 信号のオン － オフ状態を設定できます。デフォルト値は 1（オフ）で、SVON は GND にオープンです。

_8158_set_pcs_logic:

PCS 信号入力のアクティブ・ロジックを設定

_8158_set_pcs:

入力信号 PCS がオンの場合、位置のオーバーライドを可能にします。PCS のターミナル状態は「_8158_get_io_status」関数でモニタできます。

_8158_set_clr_mode

CLR 入力信号は指定したカウンタ（コマンド、位置、エラー、汎用カウンタ）をリセットできます。リセット・アクションはこの関数で設定できます。リセット・アクションには 4 つのモードがあります。詳しくは引数の説明を参照してください。

_8158_set_inp:

サーボ・ドライバのインポジション信号入力のアクティブ・ロジックを設定します。ユーザーはこの関数を使用するかどうかを選択できます。デフォルトは「使用しない」です。

_8158_set_alm:

サーボ・ドライバのアラーム信号入力のアクティブ・ロジックを設定します。アラーム信号がアクティブの場合、2 つの応答モードが使用できます。

_8158_set_erc:

この関数で ERC のロジックおよびオンタイムを設定できます。また、ERC 信号のパルス幅も設定できます。

_8158_set_erc_out:

この関数は ERC 信号を手動で出力するのに使用されます。

_8158_clr_erc:

この関数は ERC 信号がレベル・タイプの出力に指定された場合の出力をリセットするのに使用されます。

_8158_set_sd:

機械システムから SD 信号入力のアクティブ・ロジック、ラッチ制御、動作モードを設定します。ユーザーは _8158_enable_sd でこの関数を使用するかどうかを選択できます。デフォルトは「使用しない」です。

_8158_enable_sd:

SD 信号入力を使用するかどうか設定します。デフォルト設定は「使用しない」です。

_8158_set_limit_logic:

EL ロジック、常時開、常時閉を設定します。

_8158_set_limit_mode:

EL 信号の応答モードを設定します。

_8158_get_io_status:

各軸のすべての I/O ステータスを取得します。各ビットの定義は以下の通りです :

ビット	名称	説明
0	RDY	RDY ピン入力
1	ALM	アラーム信号
2	+EL	正のリミット・スイッチ
3	-EL	負のリミット・スイッチ
4	ORG	原点スイッチ
5	DIR	DIR 出力
6	EMG	EMG ステータス
7	PCS	PCS 信号入力
8	ERC	ERC ピン出力
9	EZ	インデックス信号
10	CLR	クリア信号
11	LTC	ラッチ信号入力
12	SD	減速信号入力
13	INP	インポジション信号入力
14	SVON	サーボ・オン出力ステータス

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_set_servo(I16 AxisNo, I16 on_off);
I16 _8158_set_pcs_logic(I16 AxisNo, I16 pcs_logic);
I16 _8158_set_pcs(I16 AxisNo, I16 enable);
I16 _8158_set_clr_mode(I16 AxisNo, I16 clr_mode, I16
        targetCounterInBit);
I16 _8158_set_inp(I16 AxisNo, I16 inp_enable, I16 inp_logic);
I16 _8158_set_alm(I16 AxisNo, I16 alm_logic, I16 alm_mode);
```

I16 _8158_set_erc(I16 AxisNo, I16 erc_logic, I16 erc_pulse_width, I16 erc_mode);
I16 _8158_set_erc_out(I16 AxisNo);
I16 _8158_clr_erc(I16 AxisNo);
I16 _8158_set_sd(I16 AxisNo, I16 sd_logic, I16 sd_latch, I16 sd_mode);
I16 _8158_enable_sd(I16 AxisNo, I16 enable);
I16 _8158_set_limit_logic(I16 AxisNo, U16 Logic);
I16 _8158_set_limit_mode(I16 AxisNo, I16 limit_mode);
I16 _8158_get_io_status(I16 AxisNo, U16 *io_sts);

Visual Basic (Windows 2000/XP)

B_8158_set_servo(ByVal AxisNo As Integer, ByVal on_off As Integer) As Integer
B_8158_set_pcs_logic(ByVal AxisNo As Integer, ByVal pcs_logic As Integer) As Integer
B_8158_set_pcs(ByVal AxisNo As Integer, ByVal enable As Integer)As Integer
B_8158_set_clr_mode(ByVal AxisNo As Integer, ByVal clr_mode As Integer, ByBal targetCounterInBit as Integer) As Integer
B_8158_set_inp(ByVal AxisNo As Integer, ByVal inp_enable As Integer, ByVal inp_logic As Integer) As Integer
B_8158_set_alm(ByVal AxisNo As Integer, ByVal alm_logic As Integer, ByVal alm_mode As Integer) As Integer
B_8158_set_erc(ByVal AxisNo As Integer, ByVal erc_logic As Integer, ByVal erc_pulse_width As Integer, ByVal erc_mode As Integer) As Integer
B_8158_set_erc_out(ByVal AxisNo As Integer) As Integer
B_8158_clr_erc(ByVal AxisNo As Integer) As Integer
B_8158_set_sd(ByVal AxisNo As Integer, ByVal sd_logic As Integer, ByVal sd_latch As Integer, ByVal sd_mode As Integer) As Integer
B_8158_enable_sd(ByVal AxisNo As Integer, ByVal Enable As Integer) As Integer
B_8158_set_limit_logic(ByVal AxisNo As Integer, ByVal Logic As Integer) As Integer
B_8158_set_limit_mode(ByVal AxisNo As Integer, ByVal limit_mode As Integer) As Integer
I16 _8158_get_io_status(ByVal AxisNo As Integer, io_sts As Integer) As Integer

@ 引数

AxisNo: ターゲット軸の軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

on_off: SVON 信号のオン－オフ状態

値	意味
0	オン
1	オフ

pcs_logic: PCS 信号入力のロジック

値	意味
0	負ロジック
1	正ロジック

enable: 使用するかどうかを設定

値	意味
0	使用しない
1	Enable（使用する）

clr_mode: CLR 入力クリア・モードの指定

clr_mode = 0、フォーリング・エッジでクリア（デフォルト）

clr_mode = 1、ライジング・エッジでクリア

clr_mode = 2、低レベルでクリア

clr_mode = 3、高レベルでクリア

targetCounterInBit: ビットごとでターゲット・カウンタをクリアするかどうかを設定

値	意味
ビット	説明
0	CLR 入力がオンになると、コマンド・カウンタをリセットする
1	CLR 入力がオンになると、位置カウンタをリセットする
2	CLR 入力がオンになると、エラー・カウンタをリセットする
3	CLR 入力がオンになると、汎用位置カウンタをリセットする

inp_enable: INP 機能を使用するかどうかを設定

inp_enable = 0、使用しない（デフォルト）

inp_enable = 1、使用する

inp_logic: INP 信号のアクティブ・ロジックを設定

値	意味
0	負ロジック
1	正ロジック

alm_logic: アラーム信号のアクティブ・ロジックを設定

値	意味
0	負ロジック
1	正ロジック

alm_mode: ALARM 信号を受信したときの応答モード

値	意味
0	モータを直ちに停止する（デフォルト）
1	モータを減速してから停止する

erc_logic: ERC 信号のアクティブ・ロジックを設定

値	意味
0	負ロジック
1	正ロジック

erc_pulse_width: ERC 信号のパルス幅を設定

値	意味
0	12 µs
1	102 µs
2	409 µs
3	1.6 ms
4	13 ms
5	52 ms
6	104 ms
7	レベル出力

erc_mode:

値	意味
0	使用しない
1	EL、ALM、EMG 入力で停止した場合、ERC を出力
2	原点復帰完了時 ERC を出力
3	1 と 2 の両方

sd_logic:

値	意味
0	負ロジック
1	正ロジック

sd_latch: SD 信号のラッチ制御を設定

値	意味
0	ラッチしない
1	ラッチする

sd_mode: SD 信号の応答モードを設定

値	意味
0	減速のみ
1	減速後、停止

enable: 高速フィードの下降点を設定

値	意味
0	自動設定
1	手動設定（デフォルト）

Logic: PEL/MEL のロジックを設定

値	意味
0	常時低（常時開）
1	常時高（常時閉）

limit_mode:

値	意味
0	直ちに停止
1	減速後、停止

***io_sts**: I/O ステータス。6.12 の関数の説明を参照してください。

6.13 割り込み制御

@ 名称

_8158_int_control・ INT サービスを使用するかどうかを設定

_8158_set_motion_int_factor・ モーション関連の割り込みのファクタを設定

_8158_wait_error_interrupt・ エラー関連の割り込みを待機

_8158_wait_motion_interrupt・ モーション関連の割り込みを待機

@ 説明

_8158_int_control:

この関数は Windows の割り込みイベントがホスト PC を制御可能にするのに使用されます。

_8158_set_motion_int_factor:

この関数を使えば、イベント割り込みを発生させるモーション関連ファクタを選択できます。_8158_int_control() で割り込みサービスをオンにすると、エラーをマスクすることはできません。割り込み機能が有効になると、_8158_wait_motion_interrupt() を使用して、イベントを待機できます。

_8158_wait_error_interrupt:

_8158_int_control() で割り込み機能を有効にすると、この関数を使ってエラーの割り込みを待機できます。動作理論についてはセクション 4.8 を参照してください。

_8158_wait_motion_interrupt:

_8158_int_control() で割り込み機能を有効にし、_8158_set_motion_int_factor() で割り込みファクタを設定すると、この関数を使って特定の割り込みを待機できます。この関数の実行中、イベントがトリガされるか、関数がタイムアウトになるまで、プロセスは停止しません。

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_int_control(I16 card_id, I16 intFlag);
```

I16 _8158_set_motion_int_factor(I16 AxisNo, U32 int_factor);
I16 _8158_wait_error_interrupt(I16 AxisNo, I32 TimeOut_ms);
I16 _8158_wait_motion_interrupt(I16 AxisNo, I16 IntFactorBitNo, I32 TimeOut_ms);

Visual Basic (Windows 2000/XP)

B_8158_int_control(ByVal card_id As Integer, ByVal intFlag As Integer) As Integer
B_8158_wait_error_interrupt(ByVal AxisNo As Integer, ByVal TimeOut_ms As Long) As Integer
B_8158_wait_motion_interrupt(ByVal AxisNo As Integer, ByVal IntFactorBitNo As Integer, ByVal TimeOut_ms As Long) As Integer
B_8158_set_motion_int_factor(ByVal AxisNo As Integer, ByVal int_factor As Long) As Integer

@ 引数

card_id: ターゲットの PCI-8158 カードのインデックスを指定します。card_id は DIP スイッチ（SW1）によって決定されるか、スロットのシーケンスに依存します。_8158_initial() を参照してください。

intFlag: 割り込み機能を使用するかどうかを設定

値	意味
0	使用しない
1	Enable（使用する）

AxisNo: ターゲット軸の軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

int_factor: 割り込みファクタ

モーション INT のファクタ

値	意味（0: 無効、1: 有効）
ビット	説明
0	常時停止
1	バッファ内の次のコマンドを開始
2	コマンド・プリレジスタ 2 はエンプティで、次のコマンドの書き込みが可能
3	（逆）（常に 0 に設定）
4	加速開始
5	加速終了
6	減速開始
7	減速終了
8	+ ソフト・リミットまたはコンパレータ 1 がオン
9	- ソフト・リミットまたはコンパレータ 2 がオン
10	エラー・コンパレータまたはコンパレータ 3 がオン
11	汎用コンパレータまたはコンパレータ 4 がオン
12	トリガ・コンパレータまたはコンパレータ 5 がオン
13	CLR 入力でカウンタをリセット
14	LTC 入力でカウンタをラッチ
15	ORG 入力でカウンタをラッチ
16	SD 入力がオン
17	（逆）（常に 0 に設定）
18	CSTA 入力または _8158_start_move_all() がオン
19~31	定義なし（常に 0 に設定）

TimeOut_ms: タイムアウト間隔（ms 単位）を指定します。TimeOut_ms がゼロの場合、関数は指定した対象物の状態をテストして結果を正に返します。TimeOut_ms が -1 の場合、関数のタイムアウト間隔は無効（無期限）になります。

IntFactorBitNo: INT ファクタのビット数を指定します。

例えば、IntFactorBitNo = 4 の場合、「Acceleration Start（加速開始）」の割り込みのファクタを待機します。

6.14 位置制御およびカウンタ

@ 名称

_8158_get_position・ フィードバック位置カウンタの値を取得

_8158_set_position・ フィードバック位置カウンタの設定

_8158_get_command・ コマンド位置カウンタの値を取得

_8158_set_command・ コマンド位置カウンタの設定

_8158_get_error_counter・ 位置エラー・カウンタの値を取得

_8158_reset_error_counter・ 位置エラー・カウンタのリセット

_8158_get_general_counter・ 汎用カウンタの値を取得

_8158_set_general_counter・ 汎用カウンタの設定

_8158_get_target_pos・ ターゲット位置レコーダの値を取得

_8158_reset_target_pos・ ターゲット位置レコーダのリセット

_8158_get_res_distance・ モーションから蓄積された残存パルスを取得

_8158_set_res_distance・ 残存パルス記録の設定

@ 説明

_8158_get_position:

この関数はフィードバック位置カウンタの値を表示するのに使用されます。この値は _8158_set_move_ratio() による移動比設定によってすでに処理されていることに注意してください。移動比が 0.5 の場合、位置の値は 2 倍になります。フィードバック・カウンタのソースは外部 EA/EB または 8158 の内部パルス出力になるように関数 _8158_set_feedback_src() によって設定できます。

_8158_set_position:

この関数はフィードバック位置カウンタを指定された値を変更するのに使用されます。設定される値は移動比によって処理されることに注意してください。移動比が 0.5 の場合、設定値は与えられた値の 2 倍になります。

_8158_get_command:

この関数はコマンド位置カウンタの値を表示するのに使用されます。コマンド位置カウンタのソースは 8158 のパルス出力です。

_8158_set_command:

この関数はコマンド位置カウンタの値を変更するのに使用されます。

_8158_get_error_counter:

この関数は位置エラー・カウンタの値を表示するのに使用されます。

_8158_reset_error_counter:

この関数は位置エラー・カウンタをクリアするのに使用されます。

_8158_get_general_counter:

この関数は汎用カウンタの値を表示するのに使用されます。

_8158_set_general_counter:

この関数は汎用カウンタのカウンティング・ソースを設定たり、汎用カウンタの値を変更したりするのに使用されます。(デフォルトでは、ソースはパルス入力)

_8158_get_target_pos:

この関数はターゲット位置レコーダの値を表示するのに使用されます。ターゲット位置レコーダは 8158 のソフトウェア・ドライバによって管理され、現在の動作モーションが落ち着く位置を記録します。

_8158_reset_target_pos:

この関数はターゲット位置レコーダの新しい値を設定するのに使用されます。原点復帰完了時、または関数 _8158_set_position() によって新しいフィードバック・カウンタ値を設定した場合、この関数を呼び出す必要があります。

_8158_get_res_distance:

この関数は残存距離レコーダの値を表示するのに使用されます。ターゲット位置レコーダは 8158 のソフトウェア・ドライバによって管理され、現在の動作モーションが落ち着く位置を記録します。

_8158_set_res_distance:

この関数は残存距離レコーダの値を変更するのに使用されます。

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_get_position(I16 AxisNo, F64 *Pos);
I16 _8158_set_position(I16 AxisNo, F64 Pos);
I16 _8158_get_command(I16 AxisNo, I32 *Command);
I16 _8158_set_command(I16 AxisNo, I32 Command);
I16 _8158_get_error_counter(I16 AxisNo, I16 *error);
I16 _8158_reset_error_counter(I16 AxisNo);
I16 _8158_get_general_counter(I16 AxisNo, F64 *CntValue);
I16 _8158_set_general_counter(I16 AxisNo, I16 CntSrc, F64 CntValue);
I16 _8158_get_target_pos(I16 AxisNo, F64 *T_pos);
I16 _8158_reset_target_pos(I16 AxisNo, F64 T_pos);
I16 _8158_get_res_distance(I16 AxisNo, F64 *Res_Distance);
I16 _8158_set_res_distance(I16 AxisNo, F64 Res_Distance);
```

Visual Basic (Windows 2000/XP)

```
B_8158_get_position(ByVal AxisNo As Integer, Pos As Double) As
        Integer
B_8158_set_position(ByVal AxisNo As Integer, ByVal Pos As Double)
        As Integer
B_8158_get_command(ByVal AxisNo As Integer, Cmd As Long) As
        Integer
B_8158_set_command(ByVal AxisNo As Integer, ByVal Cmd As Long)
        As Integer
B_8158_get_error_counter(ByVal AxisNo As Integer, ByRef error As
        Integer) As Integer
B_8158_reset_error_counter(ByVal AxisNo As Integer) As Integer
B_8158_set_general_counter(ByVal AxisNo As Integer, ByVal CntSrc
        As Integer, ByVal CntValue As Double) As Integer
B_8158_get_general_counter(ByVal AxisNo As Integer, ByRef Pos As
        Double) As Integer
```

B_8158_reset_target_pos(ByVal AxisNo As Integer, ByVal Pos As Double) As Integer
B_8158_get_target_pos(ByVal AxisNo As Integer, ByRef Pos As Double) As Integer
B_8158_set_res_distance(ByVal AxisNo As Integer, ByVal Res_Distance As Double) As Integer
B_8158_get_res_distance(ByVal AxisNo As Integer, ByRef Res_Distance As Double) As Integer

@ 引数

AxisNo: ターゲット軸の軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Pos, ***Pos**: フィードバック位置カウンタ値(_8158_get/ set_position)

範囲 : -134217728-134217727

Cmd, ***Cmd**: コマンド位置カウンタ値、

範囲 : -134217728-134217727

***error**: 位置エラー・カウンタ値、

範囲 : -32768-32767

CntSrc: 汎用カウンタ・ソース

値	意味
0	コマンド・パルス
1	EA/EB
2	パルス入力
3	システム・クロック÷ 2

CntValue, ***CntValue**: カウンタ値

TargetPos, ***TargetPos**: ターゲット位置レコーダ値、

範囲 : -134217728~134217727

ResDistance, ***ResDistance**: 残存距離

6.15 位置比較およびラッチ

@ 名称

_8158_set_trigger_logic・

_8158_set_trigger_comparator・ トリガ・コマンドの設定

_8158_set_error_comparator・ エラー・コンパレータの設定

_8158_set_general_comparator・ 汎用コンパレータの設定

_8158_set_latch_source・ カウンタのラッチ・タイミングの設定

_8158_set_ltc_logic・ LTC 信号のロジックを設定

_8158_get_latch_data・ カウンタからラッチ・デ―タを取得

@ 説明

_8158_set_trigger_logic:

この関数はCMP信号のロジックを設定するのに使用されます。

_8158_set_error_comparator:

この関数はエラー・コンパレータの比較方法と値を設定するのに使用されます。位置エラー・カウンタの値が比較値に達すると、8158 はホスト PC に割り込みを送信します。セクション 6.14「割り込み制御」も参照してください。

_8158_set_general_comparator:

この関数は汎用コンパレータの比較ソース・カウンタ、比較方法、値を設定するのに使用されます。比較条件に適合すると、4 つの応答のうちの 1 つが実行されます。詳しい設定方法については、引数の説明を参照してください。

_8158_set_trigger_comparator:

この関数はトリガ・コンパレータの比較ソース・カウンタ、比較方法、値を設定するのに使用されます。比較ソース・カウンタの値が比較値に達すると、8158 は CMP からパルス出力を発生し、割り込み（event_int_status、ビット 12）をホスト PC に送信します。

_8158_set_latch_source:

ラッチ・トリガ・ソースには 4 種類あります。この関数を使うと、カウンタのデータをラッチするイベント・ソースを選択できます。

_8158_set_ltc_logic:

この関数はラッチ入力のロジックを設定するのに使用されます。

_8158_get_latch_data:

この関数はラッチ信号到達後のカウンタのラッチされた値を表示するのに使用されます。

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_set_trigger_logic(I16 AxisNo, I16 Logic);
I16 _8158_set_error_comparator(I16 AxisNo, I16 CmpMethod, I16
    CmpAction, I32 Data);
I16 _8158_set_general_comparator(I16 AxisNo, I16 CmpSrc, I16
    CmpMethod, I16 CmpAction, I32 Data);
I16 _8158_set_trigger_comparator(I16 AxisNo, I16 CmpSrc, I16
    CmpMethod, I32 Data);
I16 _8158_set_latch_source(I16 AxisNo, I16 LtcSrc);
I16 _8158_set_ltc_logic(I16 AxisNo, I16 LtcLogic);
I16 _8158_get_latch_data(I16 AxisNo, I16 CounterNo, F64 *Pos);
```

Visual Basic (Windows 2000/XP)

```
B_8158_set_trigger_logic(ByVal AxisNo As Integer, ByVal Logic As
    Integer) As Integer
B_8158_set_error_comparator(ByVal AxisNo As Integer, ByVal
    CmpMethod As Integer, ByVal CmpAction As Integer, ByVal
    Data As Long) As Integer
B_8158_set_general_comparator(ByVal AxisNo As Integer, ByVal
    CmpSrc As Integer, ByVal CmpMethod As Integer, ByVal
    CmpAction As Integer, ByVal Data As Long) As Integer
B_8158_set_trigger_comparator(ByVal AxisNo As Integer, ByVal
    CmpSrc As Integer, ByVal CmpMethod As Integer, ByVal Data
    As Long) As Integer
B_8158_set_latch_source(ByVal AxisNo As Integer, ByVal LtcSrc As
    Integer) As Integer
B_8158_set_ltc_logic(ByVal AxisNo As Integer, ByVal StcLogic As
    Integer) As Integer
```

B_8158_get_latch_data(ByVal AxisNo As Integer, ByVal CounterNo As Integer, Pos As Double) As Integer

@ 引数

AxisNo: ターゲット軸の軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Logic: 比較するトリガのロジック

値	意味
0	負ロジック
1	正ロジック

CmpSrc: 比較するソース・カウンタ

値	意味
0	コマンド・カウンタ
1	フィードバック・カウンタ
2	エラー・カウンタ
3	汎用カウンタ

CmpMethod: 比較方法

値	意味
0	比較しない（無効）
1	データ = ソース・カウンタ（方向独立）
2	データ = ソース・カウンタ（カウント・アップのみ）

値	意味
3	データ = ソース・カウンタ（カウント・ダウンのみ）
4	データ > ソース・カウンタ
5	データ < ソース・カウンタ

Data: 比較値（位置）

CmpAction:

値	意味
0	アクションなし
1	直ちに停止
2	減速後、停止

ltc_src:

値	意味
0	LTC ピン入力
1	ORG ピン入力
2	汎用コンパレータの条件が適合
3	トリガコンパレータの条件が適合

ltc_logic: LTC 信号作動エッジ

値	意味
0	負ロジック
1	正ロジック

CounterNo: ラッチするカウンタを指定

値	意味
0	コマンド・カウンタ
1	フィードバック・カウンタ
2	エラー・カウンタ
3	汎用カウンタ

***Pos**: ラッチ・データ（位置）

6.16 連続モーション

@ 名称

_8158_set_continuous_move・ 絶対モーションの連続モーションを有効にする

_8158_check_continuous_buffer・ バッファがエンプティかどうかを確認

_8158_dwell_move・ タイマ移動の設定

@ 説明

_8158_set_continuous_move:

この関数は連続モーションのコマンド列の前後に必要とされます。

_8158_check_continuous_buffer:

この関数はコマンド・プリレジスタ（バッファ）がエンプティかどうかを検出するに使用されます。コマンド・プリレジスタ（バッファ）がエンプティになると、次のモーション・コマンドを書き込めます。エンプティではないと、新しいコマンドは 2 番目のコマンド・プリレジスタに直前のコマンドを上書きします。1 の応答コードはバッファに空きがないことを意味します。バッファに空きがある場合の応答コードは 0 です。

_8158_dwell_move:

この関数は、指定された時間実際のモーションを実行しないタイマ移動を開始するのに使用されます。

例 :

```
_8158_set_continuous_move( 2, 1 ); // start continuous move
_8158_start_tr_move( 2, 20000.0, 10.0, 10000.0, 0.1, 0.1);
_8158_dwell_move( 2, 2000); //dwell move for 2 sec.
_8158_start_sr_move( 2, 20000.0, 10.0, 10000.0, 0.1, 0.1, 0, 0 );
_8158_set_continuous_move( 2, 0 ); //end continuous move
```

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_set_continuous_move(I16 AxisNo, I16 Enable);
```

I16 _8158_check_continuous_buffer(I16 AxisNo);
I16 _8158_dwell_move(I16 AxisNo, F64 miniSecond);

Visual Basic (Windows 2000/XP)

B_8158_set_continuous_move(ByVal AxisNo As Integer, ByVal Enable As Integer) As Integer
B_8158_check_continuous_buffer(ByVal AxisNo As Integer) As Integer
B_8158_dwell_move(ByVal AxisNo As Integer, ByVal miniSecond As Double) As Integer

@ 引数

AxisNo: ターゲット軸の軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Enable: 連続モーション・スイッチのロジック

値	意味
0	連続モーション列終了（無効）
1	連続モーション列開始（有効）

millisecond: タイマ移動の時間、単位は ms

6.17 多軸同時動作

@ 名称

_8158_set_tr_move_all・ 多軸同時動作のセットアップ

_8158_set_ta_move_all・ 多軸同時動作のセットアップ

_8158_set_sr_move_all・ 多軸同時動作のセットアップ

_8158_set_sa_move_all・ 多軸同時動作のセットアップ

_8158_start_move_all・ 多軸台形プロファイル・モーションを開始

_8158_stop_move_all・ 多軸モーションの同時停止

@ 説明

上記関数は異なるカードの場合も含む多軸同時動作で使用します。多軸同時動作は指定した軸を同時にスタートまたは停止させる動作です。移動する軸はパラメータ「AxisArray」で指定され、軸数は _8158_set_tr_move_all() のパラメータ「TotalAxes」で定義されます。

_8158_set_xx_move_all() を使って正しくセットアップすると、指定したすべての軸は関数 _8158_start_move_all() で相対台形移動を開始し、_8158_stop_move_all() で停止します。両方の関数は指定したすべての軸のスタート / ストップのモーションが同時に開始するのを保証します。上記の 2 つの関数が必要な場合、セクション 2.6 に従って接続しなければならないことに注意してください。

以下のコードは上記関数の使用方法を示しています。このコードで、軸 0 と軸 1 は距離 80000.0 と 120000.0 にそれぞれ移動します。距離の比に比例した速度と加速度を選択すると、両方の軸はそれぞれの終点に同時に到達します。

[例]

```
I16 axes[2] = {0, 1};
F64 dist[2] = {80000.0, 120000.0},
F64 str_vel[2] = {0.0, 0.0},
F64 max_vel[2] = {4000.0, 6000.0},
F64 Tacc[2] = {0.1, 0.6},
F64 Tdec[2] = {0.1, 0.6};

_8158_set_tr_move_all(2, axes, dist, str_vel, max_vel, Tacc, Tdec);
_8158_start_move_all(axes[0]);
```

@ 文字列

C/C++(Windows 2000/XP)

I16 _8158_set_tr_move_all(I16 TotalAxes, I16 *AxisArray, F64 *DistA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA);

I16 _8158_set_ta_move_all(I16 TotalAx, I16 *AxisArray, F64 *PosA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA);

I16 _8158_set_sr_move_all(I16 TotalAx, I16 *AxisArray, F64 *DistA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA, F64 *SVaccA, F64 *SVdecA);

I16 _8158_set_sa_move_all(I16 TotalAx, I16 *AxisArray, F64 *PosA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA, F64 *SVaccA, F64 *SVdecA);

I16 _8158_start_move_all(I16 FirstAxisNo);

I16 _8158_stop_move_all(I16 FirstAxisNo);

Visual Basic (Windows 2000/XP)

B_8158_set_tr_move_all(ByVal TotalAxes As Integer, ByRef AxisArray As Integer, ByRef DistA As Double, ByRef StrVelA As Double, ByRef MaxVelA As Double, ByRef TaccA As Double, ByRef TdecA As Double) As Integer

B_8158_set_sa_move_all(ByVal TotalAxes As Integer, ByRef AxisArray As Integer, ByRef PosA As Double, ByRef StrVelA As Double, ByRef MaxVelA As Double, ByRef TaccA As Double, ByRef TdecA As Double, ByRef SVaccA As Double, ByRef SVdecA As Double) As Integer

B_8158_set_ta_move_all(ByVal TotalAxes As Integer, ByRef AxisArray As Integer, ByRef PosA As Double, ByRef StrVelA As Double, ByRef MaxVelA As Double, ByRef TaccA As Double, ByRef TdecA As Double) As Integer

B_8158_set_sr_move_all(ByVal TotalAxes As Integer, ByRef AxisArray As Integer, ByRef DistA As Double, ByRef StrVelA As Double, ByRef MaxVelA As Double, ByRef TaccA As Double, ByRef TdecA As Double, ByRef SVaccA As Double, ByRef SVdecA As Double) As Integer

B_8158_start_move_all(ByVal FirstAxisNo As Integer) As Integer

B_8158_stop_move_all(ByVal FirstAxisNo As Integer) As Integer

@ 引数

TotalAxes: 同時モーションの軸数

***AxisArray**: 移動するように指定された軸番号の配列

***DistA**: 指定された距離の配列（単位 : パルス）

***StrVelA**: 初速度の配列（単位 : パルス / 秒）

***MaxVelA**: 最大速度の配列（単位 : パルス / 秒）

***TaccA**: 加速時間の配列（単位 : 秒）

***TdecA**: 減速時間の配列（単位 : 秒）

***PosA**: 指定された位置の配列（単位 : パルス）

***SvaccA**: S 曲線加速が実行される指定された速度間隔の配列

***SvdecA**: S 曲線減速が実行される指定された速度間隔の配列

FirstAxisNo: AxisArray の最初のエレメント

6.18 汎用 DIO

@ 名称

_8158_set_gpio_output・ デジタル出力の設定

_8158_get_gpio_output・ デジタル出力の取得

_8158_get_gpio_input・ デジタル入力の取得

_8158_set_gpio_input_function ・ デジタル入力の信号タイプを設定

@ 説明

_8158_set_gpio_output:

PCI-8158 は 8 つのデジタル出力チャンネルを備えています。この関数を使えば、デジタル出力を制御できます。

_8158_get_gpio_output:

この関数はデジタル出力のステータスを取得するのに使用されます。

_8158_get_gpio_input:

PCI-8158 は 8 つのデジタル入力チャンネルを備えています。この関数を使えば、デジタル出力のステータスを取得できます。

_8158_set_gpio_input_function:

PCI-8158 は 8 つのデジタル入力チャンネルを備えています。この関数を使えば、複数の入力信号の 1 つを特定の DI チャンネルに設定できます。入力信号には、LTCn、SDn、PCSn、CLRn、EMG が含まれます。（n は軸のインデックス）

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_set_gpio_output(I16 card_id, I16 DoValue);
I16 _8158_get_gpio_output(I16 card_id, I16 * DoValue);
I16 _8158_get_gpio_input(I16 card_id, I16 * DiValue);
I16 _8158_set_gpio_input_function(I16 card_id, I16 Channel, I16Select,
        I16 Logic);
```

Visual Basic (Windows 2000/XP)

B_8158_set_gpio_output(ByVal card_id As Integer, ByVal DoValue As Integer) As Integer

B_8158_get_gpio_output(ByVal card_id As Integer, DoValue As Integer) As Integer

B_8158_get_gpio_input(ByVal card_id As Integer, DiValue As Integer) As Integer

B_8158_set_gpio_input_function(ByVal card_id As Integer, ByVal Channel As Integer, ByVal Select As Integer, ByVal Logic As Integer)As Integer

@ 引数

card_id: PCI-8158 カードのインデックスを指定します。card_id は DIP スイッチ（SW1）によって決定されるか、スロットのシーケンスに依存します。_8158_initial() を参照してください。

DoValue, ***DoValue**: デジタル出力値。Bit 0-7: D_out0-7。

***DiValue**: デジタル入力値、Bit 0-7: D_in0-7

Channel: デジタル・チャンネル DI0 – DI7

Select: 信号タイプの選択

値	意味
0	汎用 DI（デフォルト）
1	LTC（アクティブ低）
2	SD（アクティブ低）
3	PCS（アクティブ低）
4	CLR（アクティブ低）
5	EMG（アクティブ低）

Logic: 入力信号のロジック

値	意味
0	逆にしない（デフォルト）

値	意味
1	逆

6.19 ソフト・リミット

@ 名称

_8158_disable_soft_limit・ ソフト・リミット機能を無効にする

_8158_enable_soft_limit・ ソフト・リミット機能を有効にする

_8158_set_soft_limit・ ソフト・リミットの設定

@ 説明

_8158_disable_soft_limit:

この関数はソフト・リミット機能を無効にするのに使用されます。

_8158_enable_soft_limit:

この関数はソフト・リミット機能を有効にするのに使用されます。同機能を有効にすると、ソフト・リミットのアクションは実際のリミットと全く同じになります。

_8158_set_soft_limit:

この関数はソフト・リミット値を設定するのに使用されます。

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_disable_soft_limit(I16 AxisNo);
I16 _8158_enable_soft_limit(I16 AxisNo, I16 Action);
I16 _8158_set_soft_limit(I16 AxisNo, I32 PlusLimit, I32 MinusLimit);
```

Visual Basic (Windows 2000/XP)

```
B_8158_disable_soft_limit(ByVal AxisNo As Integer) As Integer
B_8158_enable_soft_limit(ByVal AxisNo As Integer, ByVal Action As
        Integer) As Integer
B_8158_set_soft_limit(ByVal AxisNo As Integer, ByVal PlusLimit As
        Long, ByVal MinusLimit As Long) As Integer
```

@ 引数

AxisNo: ターゲット軸の軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Action: ソフト・リミットの応答方法

値	意味
0	INT のみ
1	直ちに停止
2	減速後、停止

PlusLimit: ソフト・リミット値、正方向

MinusLimit: ソフト・リミット値、負方向

6.20 バックラッシュ補正 / 振動抑制

@ 名称

_8158_backlash_comp・ バックラッシュ調整パルスを補正に設定

_8158_suppress_vibration・ 振動抑制時間の設定

_8158_set_fa_speed・ FA 速度の設定

@ 説明

_8158_backlash_comp:

方向が変更されると、8158 はコマンドを送信する前に、バックラッシュ調整パルスを出力します。この関数は補正パルス数を設定するのに使用されます。

_8158_suppress_vibration:

この関数は、コマンド移動完了直後に負方向のパルスと正方向のパルスを 1 つずつ出力することで、機械システムの振動を抑制するのに使用されます。

_8158_set_fa_speed:

この関数はバックラッシュ補正またはスリップ補正のための低速度を指定するのに使用されます。また、原点復帰動作のための逆の低速度としても使用されます。

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_backlash_comp(I16 AxisNo, I16 CompPulse, I16 Mode);
I16 _8158_suppress_vibration(I16 AxisNo, U16 ReverseTime,
U16 ForwardTime);
I16 _8158_set_fa_speed(I16 AxisNo, F64 FA_Speed);
```

Visual Basic (Windows 2000/XP)

```
B_8158_backlash_comps (ByVal AxisNo As Integer, ByVal CompPulse
        As Integer, ByVal Mode As Integer) As Integer
B_8158_suppress_vibration(ByVal AxisNo As Integer, ByVal
        ReverseTime As Integer, ByVal ForwardTime As Integer) As
        Integer
```

B_8158_set_fa_speed(ByVal AxisNo As Integer, ByVal FA_Speed As Double) As Integer

@ 引数

AxisNo: ターゲット軸の軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

CompPulse: 指定された補正パルス数、12 ビット

モード :

値	意味
0	電源オフ
1	バックラッシュ補正を有効にする
2	スリップ補正

ReverseTime: 指定された逆進時間、0 – 65535、単位 1.6 us

ForwardTime: 指定された前進時間、0 – 65535、単位 1.6 us

FA_Speed: FA 速度（単位 : パルス / 秒）

6.21 速度プロファイルの計算

@ 名称

_8158_get_tr_move_profile・ 相対台形速度プロファイルを取得

_8158_get_ta_move_profile・ 絶対台形速度プロファイルを取得

_8158_get_sr_move_profile・ 相対 S 曲線速度プロファイルを取得

_8158_get_sa_move_profile・ 絶対 S 曲線速度プロファイルを取得

@ 説明

_8158_get_tr_move_profile:

この関数は相対台形速度プロファイルを取得するのに使用されます。この関数を使えば、実行前に実際の速度プロファイルを取得できます。

_8158_get_ta_move_profile:

この関数は絶対台形速度プロファイルを取得するのに使用されます。この関数を使えば、実行前に実際の速度プロファイルを取得できます。

_8158_get_sr_move_profile:

この関数は相対S曲線速度プロファイルを取得するのに使用されます。この関数を使えば、実行前に実際の速度プロファイルを取得できます。

_8158_get_sa_move_profile:

この関数は絶対S曲線速度プロファイルを取得するのに使用されます。この関数を使えば、実行前に実際の速度プロファイルを取得できます。

@ 文字列

C/C++(Windows 2000/XP)

```
I16 _8158_get_tr_move_profile(I16 AxisNo, F64 Dist, F64 StrVel, F64
        MaxVel, F64 Tacc, F64 Tdec, F64 *pStrVel, F64 *pMaxVel, F64
        *pTacc, F64 *pTdec, F64 *pTconst );
I16 _8158_get_ta_move_profile(I16 AxisNo, F64 Pos, F64 StrVel, F64
        MaxVel, F64 Tacc, F64 Tdec, F64 *pStrVel, F64 *pMaxVel, F64
        *pTacc, F64 *pTdec, F64 *pTconst );
```

I16 _8158_get_sr_move_profile(I16 AxisNo, F64 Dist, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec,F64 *pStrVel, F64 *pMaxVel, F64 *pTacc, F64 *pTdec, F64 *pSVacc, F64 *pSVdec, F64 *pTconst);

I16 _8158_get_sa_move_profile(I16 AxisNo, F64 Pos, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec,F64 *pStrVel, F64 *pMaxVel, F64 *pTacc, F64 *pTdec, F64 *pSVacc, F64 *pSVdec, F64 *pTconst);

Visual Basic (Windows 2000/XP)

B_8158_get_tr_move_profile(ByVal AxisNo As Integer, ByVal Dist As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByRef pStrVel As Double, ByRef pMaxVel As Double, ByRef pTacc As Double, ByRef pTdec As Double, ByRef pTconst As Double) As Integer

B_8158_get_ta_move_profile(ByVal AxisNo As Integer, ByVal Pos As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByRef pStrVel As Double, ByRef pMaxVel As Double, ByRef pTacc As Double, ByRef pTdec As Double, ByRef pTconst As Double) As Integer

B_8158_get_sr_move_profile(ByVal AxisNo As Integer, ByVal Dist As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double, ByRef pStrVel As Double, ByRef pMaxVel As Double, ByRef pTacc As Double, ByRef pTdec As Double, ByRef pSVacc As Double, ByRef pSVdec As Double, ByRef pTconst As Double) As Integer

B_8158_get_sa_move_profile(ByVal AxisNo As Integer, ByVal Pos As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double, ByRef pStrVel As Double, ByRef pMaxVel As Double, ByRef pTacc As Double, ByRef pTdec As Double, ByRef pSVacc As Double, ByRef pSVdec As Double, ByRef pTconst As Double) As Integer

@ 引数

AxisNo: ターゲット軸の軸番号

card_id	実際の軸	AxisNo
0	0	0
	1	1
	…	…
	7	7
1	0	8
	1	9
	…	…

Dist: 指定された相対距離（単位 : パルス）

Pos: 指定された絶対位置（単位 : パルス）

StrVel: 初速度（単位 : パルス / 秒）

MaxVel: 最大速度（単位 : パルス / 秒）

Tacc: 加速時間（単位 : 秒）

Tdec: 減速時間（単位 : 秒）

SVacc: 加速時の S 曲線エリア（単位 : パルス / 秒）

注意 : 純粋な S 曲線では SVacc = 0。詳しくはセクション 4.2.4 を参照

SVdec: 減速時の S 曲線エリア（単位 : パルス / 秒）

注意 : 純粋な S 曲線では SVdec = 0。詳しくはセクション 4.2.4 を参照

*pStrVel: 計算上の初速度

*pMaxVel: 計算上の最大速度

*pTacc: 計算上の加速時間

*pTdec: 計算上の減速時間

*pSVacc: 計算上の加速時の S 曲線エリア

*pSVdec: 計算上の減速時の S 曲線エリア

*pTconst: 一定速度時間（最大速度）

6.22 応答コード

応答コードは「8158_err.h」で定義されます。意味は以下の表の通りです。

コード	意味
0	エラーなし
-10000	カード番号エラー
-10001	OS バージョン・エラー
-10002	エラーコードの ID 衝突
-10300	他のタスクを処理中
-10301	カードがありません。
-10302	ドライバを開けません。
-10303	ID マッピング・エラー
-10304	トリガ・チャンネル・エラー
-10305	トリガ・タイプ・エラー
-10306	イベントはすでに有効になっています。
-10307	イベントはまだ有効になっていません。
-10308	オンボード FIFO に空きがありません。
-10309	未知のコマンド・タイプ
-10310	未知のチップ・タイプ
-10311	カードが初期化されていません。
-10312	位置が範囲外です。
-10313	モーションがビジーです。
-10314	速度エラー
-10315	減速点エラー
-10316	軸範囲エラー
-10317	比較パラメータ・エラー
-10318	比較方法が無効です。
-10319	軸はすでに停止しています。
-10320	軸 INT 待機エラー
-10321	ユーザー・コードを書き込めません。
-10322	配列サイズが制限を超えています。
-10323	ファクタ番号エラー
-10324	範囲を有効にできません。
-10325	自動加速時間エラー
-10326	タイマ時間エラー
-10327	タイマ距離エラー
-10328	新しい位置を設定できません。
-10329	モーションが動作していません。

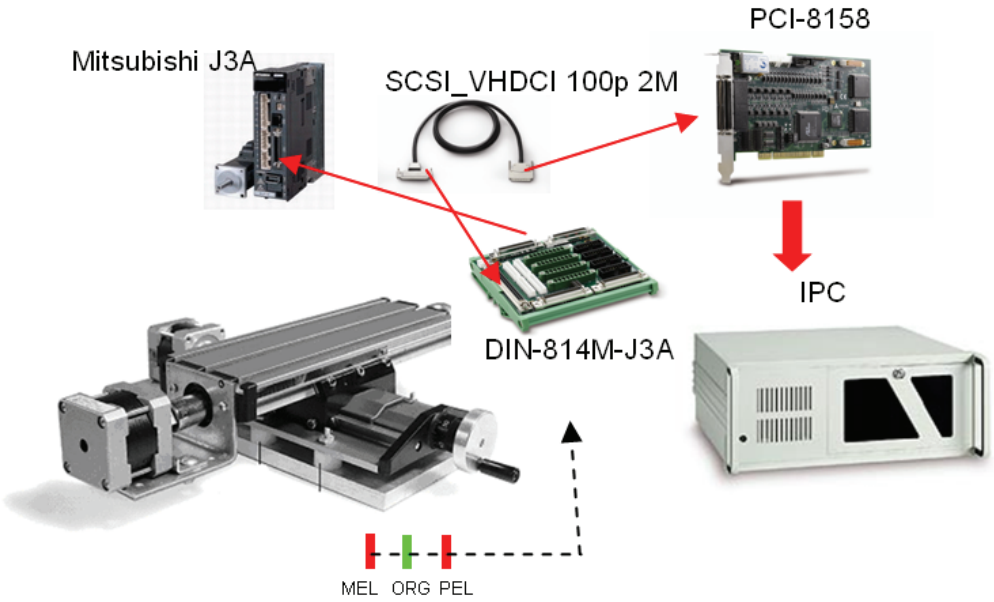
コード	意味
-10330	速度変更時間エラー
-10331	速度ターゲット・エラー
-10332	速度の割合に問題があります。
-10333	位置逆進変更エラー
-10334	カウンタ番号エラー
-10335	GPIO 入力関数パラメータ・エラー
-10336	チャンネル番号エラー
-10337	ERC モード・エラー
-10338	セキュリティ・コード・エラー

7 接続例

この章では PCI-8158、サーボ・ドライバ、ステッピング・ドライバの接続例を紹介します。

7.1 配線の概要

PCI-8158 とパルス入力サーボ・ドライバまたはステッピング・ドライバの接続は主要な接続となります。PCI-8158 と DIN-814M-J3A の接続方法は下図の通りです。



7.2 端子ボードのユーザー・ガイド

端子ボードの個別のユーザー・ガイドを参照してください。対応している端子ポートは以下の通りです :

Mitsubishi J2 Super	DIN-814M
Mitsubishi J3A	DIN-814M-J3A
Yaskawa Sigma II	DIN-814Y
Panasonic MINAS A4	DIN-814P-A4

保証方針

ADLINK をお選びくださりありがとうございました。お客様の権利を理解し、ADLINK が提供するアフターサービスを利用できるように、以下の内容を注意深くお読みください。

1. ADLINK 製品を使用する前に、ユーザーマニュアルをお読みになり、指示に正しく従ってください。損傷のある製品を返却する場合は、RMA 申込書を以下のサイトからダウンロードし、必要事項を記入して添付してください：http://rma.adlinktech.com/policy/
2. すべての ADLINK 製品の保証期間は 2 年間ですが、中国で販売された製品の保証期間は 1 年間です：
   - 保証期間は製品が ADLINK の工場から出荷された日付から始まります。
   - ADLINK が製造したのではない周辺機器および他社製品には最初の製造元の保証が適用されます。
   - ストレージ・デバイス（ハードディスクやフラッシュカードなど）を含む製品を返却する場合はデータのバックアップを取ってください。ADLINK はデータ損失の責任を負いません。
   - ADLINK のシステムでは必ずライセンスを正しく取得したソフトウェアを使用してください。ADLINK は海賊版ソフトウェアの使用を禁止しており、そのようなソフトウェアを使用したシステムにサービスを提供することはありません。ADLINK はユーザーがライセンスのないソフトウェアをインストールした製品に対してはいかなる法的責任も負いません。
   - 一般的修理では、周辺アクセサリ製品は取り外してください。周辺機器を含める必要がある場合は、送付したアイテムを RMA 要求確認用紙に必ず明記してください。ADLINK は RMA 要求確認用紙に記載されていないアイテムに対して責任を負いません。

3. 以下の条件に該当する場合、修理サービスに ADLINK の保証は適用されません :
   - ユーザーマニュアルの指示に従わないで生じた損傷
   - 製品の輸送中にユーザーの過失で生じた損傷
   - 火災、地震、洪水、落雷、汚染などの自然現象または変圧器の不正な使用による損傷
   - 不適切な保存環境（例えば、高温、高湿度、揮発性化学物質など）によって生じた損傷
   - お客様 / ユーザーがバッテリの交換中または交換後にバッテリの液漏れによって生じた損傷
   - ADLINK の技術者以外の無資格者による不正な修理によって生じた損傷
   - シリアル番号が変更または損傷された製品はサービスの対象になりません。
   - この保証の移譲または延期はできません。
   - その他の製品に ADLINK の保証は適用されません。
4. 損傷した製品を ADLINK または販売店に返却する場合、郵送費はお客様が負担するものとします。
5. 製品修理の期間短縮と品質向上のため、ADLINK のウェブサイト（http://rma.adlinktech.com/policy）から RMA 申込書をダウンロードしてください。RMA 用紙が添付されている製品の修理は優先されます。

ご質問があれば、ADLINK の FAE スタッフ（以下のアドレス）に e メールでお問い合わせください :
service@adlinktech.com.