Hivertec CPD Series

チュートリアル

2023-12-26 版　　株式会社ハイバーテック 発行

注意事項

保証範囲

1. 本製品の保証期間は、お買い上げ頂いた日より 3 年間です。保証期間中に弊社の判断により欠陥が判明した場合には、本製品を弊社に引き取り、修理または交換を行います。
2. 保証期間内外に関わらず、弊社製品の使用、供給 (納期) または故障に起因する、お客様及び第三者が被った、直接、間接、二次的な損害あるいは、遺失利益の損害に付いて、弊社は本製品の販売価格以上の責任を負わないものとしますので、予めご了承ください。

免責事項

1. 本書に記載された内容に沿わない、製品の取付、接続、設定、運用により生じた損害に対しましては、一切の責任を負いかねますので、予めご了承ください。
2. 本製品に添付のソフトウェアの実行、サンプルコード、プログラミングガイドなどの内容で作成されたプログラム（弊社で作成されたもの、お客様で作成されたものの両方）の動作により生じた損害にたいしましては、一切の責任を負いかねますので、予めご了承ください。
3. 本製品は、一般電子機器用 (工作機械・計測機器・FA/OA 機器・通信機器等) に製造された半導体製品を使用していますので、その誤作動や故障が直接、生命を脅かしたり、身体・財産等に危害を及ぼしたりする恐れのある装置 (医療機器・交通機器・燃焼機器・安全装置等) に適用できるような設計、意図、または、承認、保証もされていません。ゆえに本製品の安全性、品質および性能に関しては、本書 (またはカタログ) に記載してあること以外は明示的にも黙示的にも一切保証するものではありませんので、予めご了承ください。
4. 保証期間内外に関わらず、お客様が行った弊社の承認しない製品の改造または、修理が原因で生じた損害に対しましては、一切の責任を負いかねますので、予めご了承ください。
5. 本書に記載された内容について、弊社もしくは、第三者の特許権、著作権、商標権、その他の知的所有権の権利に対する保証または実施権の許諾を行うものではありません。また本書に記載された情報を使用したことにより第三者の知的所有権等の権利に関わる問題が生じた場合、弊社は、その責任を負いかねますので、予めご了承ください。
6. 添付ソフトウェア、ソフトウェア製品は弊社の判断により修正、機能追加などの改変をおこなう場合があります。

安全にお使い頂くために

この度は、弊社製品をご採用頂きまして、誠に有り難う御座います。本書は、本製品をご使用して頂く場合の取扱い、留意点に付いて記入してありますので、必ずご一読の上ご利用をお願い致します。

尚、本書は、本書が添付された製品の常設箇所付近の分かりやすい場所に常時保管し、必要に応じて適宜参照・確認頂きますよう、お願い致します。

安全上の注意

本製品のご使用前に、必ずこのユーザーズマニュアル及び付属書類を全て熟読し、内容を理解してから正しくご使用下さい。本製品の知識、安全の情報及び注意事項の全てに付いて習熟してからご使用下さい。

本ユーザーズマニュアルでは、安全注意事項のランクを「警告」、「注意」として区分してあります。

警 告

この表示を無視して、誤った取扱いをすると、人が死亡または重傷を負う可能性が想定される内容を示しています。

注 意

この表示を無視して、誤った取扱いをすると、人が傷害を負う可能性または物的損害が想定される内容を示しています。

対象ユーザー

注 意

本製品およびマニュアルは、以下の様なユーザーを対象としています。
・拡張用ボードの増設および配線に付いて基本的な知識を有している方
・制御用電子機器およびパソコン等に付いて基本的な知識を有している方
・OS の操作およびソフトウェア開発環境に付いて基本的な知識を有している方

添付ソフトウェア適合 OS

注 意

デバイスドライバ、ドライバ関数等の添付ソフトウェアは、Windows11 においてボードの制御を行う為のソフトウェアです。上記以外の OS でのご使用については、弊社営業までお問合せ下さい。

動かしてみるプログラム

注 意

・本製品に添付される「動かしてみる」プログラムは、ボードが正しく設定・装着されているか、動作環境が正しく設定されているかを確認するとともに、ボードの機能・動作を理解して頂く為のものです。故に使用される機器毎に固有な安全対策処理等を含んでいませんので、「動かしてみる」プログラムを定常的に機器運転に使用しないで下さい。
・モータや装置を接続して動作させる場合は、モータや装置の特性を考慮した動作条件を設定願います。特に試運転時は、十分に安全な値で実施し、徐々に所定の値に変更することをお勧めします。
・動かしてみるプログラムを使用し装置を動作させるとき、最初は速度の低いところで、また機械系に合った設定を行って動作を確認して下さい。機械系に合わない設定で動作を行うと思わぬ動きをすることがあります。

サンプルプログラム

注 意

・本製品に添付されるサンプルプログラムは、ボードを制御する手順・制御プログラムの作成方法を理解して頂く為のものです。故に使用される機器毎に固有な安全対策処理等を含んでいませんので、サンプルプログラムを定常的に機器運転に使用しないで下さい。
・モータや装置を接続して動作させる場合は、モータや装置の特性を考慮した動作条件を設定願います。特に試運転時は、十分に安全な値で実施し、徐々に所定の値に変更することをお勧めします。
・サンプルプログラムを使用し装置を動作させるとき、最初は速度の低いところで、また機械系に合った設定を行って動作を確認して下さい。機械系に合わない設定で動作を行うと思わぬ動きをすることがあります。

サンプルプログラム作成ツール

警 告

・サンプルプログラムは Microsoft Visual Studio のプロジェクト及びソースコードを添付しています。
・マイクロソフト社製品サポートのライフサイクル期間が終了した Microsoft Visual Studio の各バージョンについては、マイクロソフト社製品サポートのライフサイクル期間に確認したものであり、本マニュアル発行時点での動作を保証するものではありません。

ユーザープログラム

注 意

・本製品を使用し装置を動作させる際には、プログラムのデバッグを充分行ってから動作させて下さい。プログラムに間違いがありますと、思わぬ動きをすることがあります。
・本製品に添付されるサンプルプログラムまたはマニュアル内のコード例は、本製品のソフトウェア・ボードの機能・動作を理解して頂く為のものです。故に使用される機器毎に固有な安全対策処理・エラー処理・例外処理・排他処理等は省略されています。実際にプログラムを作成する場合は、十分に安全対策等を考慮し、必要な処理を追加してください。

試運転・調整

注 意

・本シリーズ製品を使用し装置を動作させる時は、プログラムのデバッグを充分行ってから動作させてください。プログラムに間違いがありますと、思わぬ動きをすることがあります。
・本シリーズ製品に添付してあるサンプルプログラムを使用し装置を動作させる時、最初は速度の低いところで、また機械系に合った設定を行って動作を確認してください。機械系に合わない設定で動作を行うと思わぬ動きをすることがあります。

Wi-Fi

警 告

Wi-Fi による接続は電波の受信状態が刻々と変化するため回線が切断される恐れがあります。そのため一時的にでも操作不可になると損害が発生したり危険を及ぼす可能性がある機器、製品に対してはご使用にならないでください。もしご使用になる場合も機器を安全に停止させる手段を講じてご使用ください。

はじめに

このたびは、弊社製品をご採用頂きまして、誠にありがとうございます。

本書は、CPD シリーズ製品の基本的な動作や機能を使用するための制御方法およびプログラミング方法を説明したものです。

本書の対象は、製品を使用した制御プログラミングを作成する方であり、以下の作業を終えている前提です。

- ドライバインストールし、動作確認用ソフトウェア「動かしてみる」で動作確認を終えている。
- 添付のマニュアルを全て読んでおり、機能や動作を理解しているかつ安全に動作確認やプログラミングできる環境下にある。

本書では、プログラミングする上で必要な基本的な制御方法を紹介し、プログラミングガイドとして Windows 版ライブラリ関数を使ったプログラム方法を記載しています。

物理的な接続方法やご使用になられる機構やドライバとの接続は別紙のハードウェアマニュアルをご参照ください。

また、動作や機能の詳細についてはユーザーズマニュアルをご参照ください。

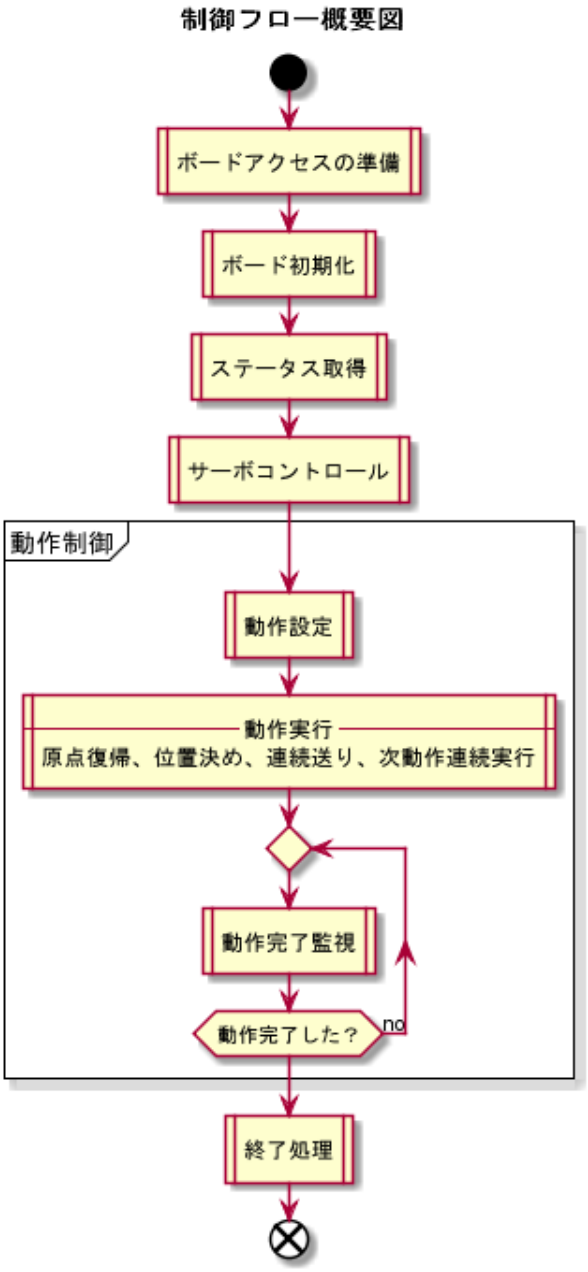
製品の全ての機能の実装方法を紹介していません。動作や機能を使用したサンプルプログラムがご用意しております。

目次

第１章

基本制御フロー解説

以下の図は、基本的な制御フロー図です。



▲図 1.1　基本制御フロー

1.1　ボードアクセスの準備

CPD を制御する場合、OS ごとのデバイス検出・リソース取得処理をしてボードにアクセスする準備をします。
Windows ライブラリ関数を用いたボードアクセス手順は次のようになります。

1. アクセスしたいボードのプロダクト ID を指定してライブラリを初期化します。
2. ボードオープンするには、オープンするボードの「設定用ロータリースイッチの値 (ボード ID)」が必要なため、指定したボードの枚数とボード ID を取得します。
3. アクセスしたいボードのプロダクト ID とボード ID を指定してオープンし、デバイスハンドルを取得します。ただしボード ID が重複している場合はオープンすることができません。
4. 以降、このハンドルを指定して制御します。

▼リスト 1.1　ボードアクセスの準備_コード例

```
ULONG nDeviceCount = 0;
ULONG hDeviceID = -1; // デバイス識別子
ULONG nBoardID[16];     // ボードID

// プロダクトIDごとに初期化
ULONG ulRet = HVT_InitUtyLib(PRODUCT_HPCI_CPD534);
if (ulRet) {
        return -1;
}

// 指定したボードの枚数とボードID取得
ulRet = CPDLib_DetectBoards(&nDeviceCount, nBoardID, PRODUCT_HPCI_CPD534);
if (ulRet) {
        return -1;
}

// 1枚目のボードをオープン
ulRet = CPDLib_Open(&hDeviceID, nBoardID[0], PRODUCT_HPCI_CPD534);
if (ulRet) {
        return -1;
}
```

※ボード ID は 0〜15 まで割り振ることができるため、最大 16 枚まで操作可能です。デバイス数を変えてもプログラムを変える必要がないように、ボード ID は 16 個分の配列を用意しておくことをおすすめします。

1.2　ボード初期化

1.2.1　レジスタ初期設定

電源投入後は全てのレジスタ値が 0 になっているので、必要な初期設定値をレジスタとオプションポートに書き込みます。

初期化の内容は CPD シリーズのユーザーズマニュアルをご参照ください。

※ソフトウェアリセットコマンドを実行した時も同様にレジスタの値が全て 0 になるので、初期設定を行うようにしてください。

Windows ライブラリ関数では、取得したデバイスハンドルを指定して初期化関数をコールします。

▼リスト 1.2　デバイス初期設定_コード例

```
        ULONG ulRet = CPDLib_SetInitSetting(hDeviceID);
        if (ulRet) {
                return -1;
        }
```

1.2.2　出力パルス形式設定

使用するモータのパルス出力にあった形式を選択します。出力パルス形式が個別パルス方式の場合は、初期化時にすでに設定されているのでこの箇所は省いても問題ありません。

Windows ライブラリ関数では、取得したデバイスハンドルを指定して設定関数をコールします。

▼リスト 1.3　出力パルス形式設定_コード例

```
        // 出力パルス形式設定
        CPDLib_SelectCmdPulse(hDeviceID, PCL_NUM_1, SELAX_ALL, DISCRETE_PULSE);
```

1.2.3　サーボ I/F の設定

モータアンプの設定や仕様に応じて設定します。使用しない入力端子はオープンにして、入力極性を A 接にしてください。

Windows ライブラリ関数では、取得したデバイスハンドルを指定して設定関数をコールします。

▼リスト 1.4　サーボ I/F の設定_コード例

```
        // 全軸SVALM : 即停止、極性A接
        CPDLib_SetSvalm(hDeviceID, PCL_NUM_1, SELAX_ALL, 0, 1);
```

1.2.4　マシン I/F の設定

接続されるセンサやアプリケーションの仕様に応じて設定します。使用しない入力端子はオープンにして、入力極性を A 接にしてください。

Windows ライブラリ関数では、取得したデバイスハンドルを指定して設定関数をコールします。

▼リスト 1.5　マシン I/F の設定_コード例

```
        // 全軸ELS : 検出時即停止、極性A接
        CPDLib_SetEls(hDeviceID, PCL_NUM_1, SELAX_ALL, 0, 1);

        // 全軸DLS : 検出時減速、ラッチしない、極性A接
        CPDLib_SetDls(hDeviceID, PCL_NUM_1, SELAX_ALL, 0, 0, 1);

        // 全軸OLS : 極性A接
        CPDLib_SetOls(hDeviceID, PCL_NUM_1, SELAX_ALL, 1);
```

1.3　サーボコントロール

サーボモータを制御対象としている場合はサーボオンしてから動作を開始します。サーボ ON せずにスタートコマンドを実行してもエラーにはならずパルスは出力されます。

Windows ライブラリ関数では、取得したデバイスハンドルを指定してコントロール関数をコールします。

▼リスト 1.6　サーボコントロール_コード例

```
// 全軸サーボON
CPDLib_CtrlSvon(hDeviceID, PCL_NUM_1, SELAX_ALL, TRUE);
```

1.4 原点復帰

装置の可動部の位置の管理は通常「原点復帰完了位置」を原点とする座標系上の位置で管理します。そのため、どの動作をするにしてもまず原点復帰を行う必要があります。

原点復帰の動作モードは機構や設計に合わせて適切なものを選択する必要がありますが、どのような開始点でも原点を探すことができる「原点サーチ」を推奨しています。原点サーチの動作手順は以下の通りです。

1. 速度波形設定：速度波形に関わるレジスタの設定を行います。
2. 原点復帰方法の設定
3. 原点復帰実行

Windows ライブラリ関数では、取得したデバイスハンドルを指定して設定関数と原点サーチ関数をコールします。

▼リスト 1.7　原点サーチ_コード例

```
// 速度設定
CPD_SPEED CpdSpd;
CpdSpd.prmg = 299;
CpdSpd.prfl = 3000;
CpdSpd.prfh = 5000;
CpdSpd.prur = 1364;
CpdSpd.prdr = 0;
CpdSpd.prus = 0;
CpdSpd.prds = 0;
CpdSpd.prdp = 0;
CpdSpd.dptype = 0;

// 全軸に設定する
CPDLib_SetSpeed(hDeviceID, PCL_NUM_1, SELAX_ALL, &CpdSpd);

// 抜け出し量設定
CPDLib_SetPosition(hDeviceID, PCL_NUM_1, SELAX_ALL, 500);

// 原点復帰方法 : OLS ON
CPDLib_SetOriginOpeMode(hDeviceID, PCL_NUM_1, SELAX_ALL, ORM_0);

// 原点サーチ : 正方向にFH定速スタート
CPDLib_MoveOriginSearch(hDeviceID, PCL_NUM_1, SELAX_ALL, EXECMD_STAFH, DIRECTION_POSITIVE);
```

1.5 ステータス取得

各種ステータスを取得してボードの状態や接続された機器の状態を確認します。

Windows ライブラリ関数では、取得したデバイスハンドルを指定してステータス読み出し関数をコールします。

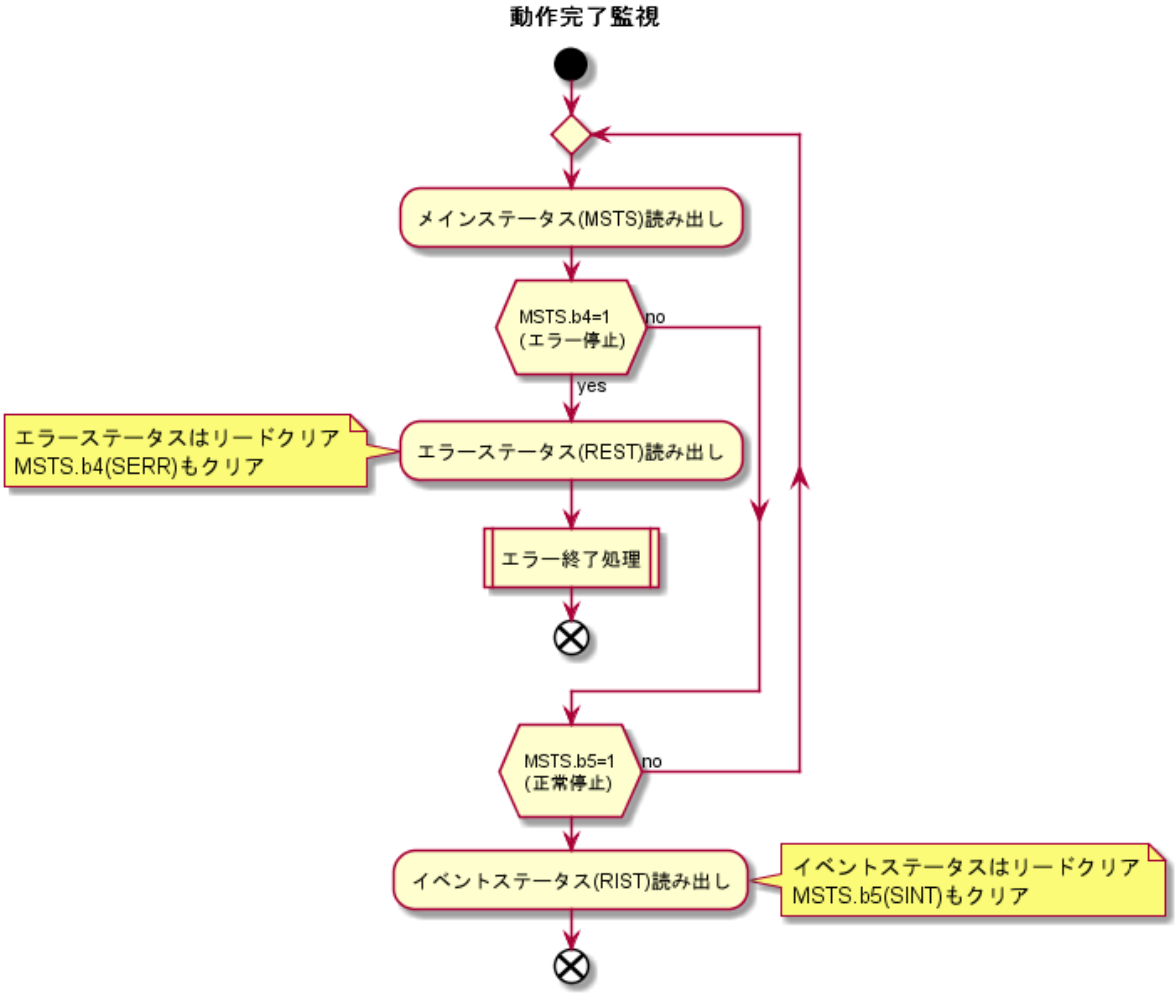
▼リスト 1.8　ステータス取得_コード例

```
// 全軸ステータス取得
WORD msts[4], ssts[4], vel[4];
ULONG ests[4], ists[4];
LONG ctr1[4], ctr2[4];
CPDLib_ReadMsts(hDeviceID, PCL_NUM_1, SELAX_ALL, msts); // メインステータス
CPDLib_ReadSsts(hDeviceID, PCL_NUM_1, SELAX_ALL, ssts); // サブステータス
CPDLib_ReadCtr(hDeviceID, PCL_NUM_1, SELAX_ALL, CTR1_NUM, ctr1); // 指令位置
CPDLib_ReadCtr(hDeviceID, PCL_NUM_1, SELAX_ALL, CTR2_NUM, ctr2); // エンコーダ位置
CPDLib_ReadVelocity(hDeviceID, PCL_NUM_1, SELAX_ALL, vel);      // 現在速度
```

サブステータスは、機構のマシン I/F、サーボ I/F の状態によって読み取れる値が変わります。初期状態がご使用になられる機構の状態と整合性が取れているか確認してください。

1.6　動作完了監視

以下は動作完了を監視するときのステータス取得フローです。



▲図 1.2　動作完了監視

エラー・イベントステータスは読み出しクリアであるため毎回読み出さず、メインステータスのエラー・イベント発生ビットが立っていた場合に取得して保持しておきます。

Windows ライブラリ関数では、取得したデバイスハンドルを指定してステータス読み出し関数をコールします。

▼リスト 1.9　動作完了監視_コード例

```
        BYTE axis_bit[4] = { SELAX_X , SELAX_Y , SELAX_Z , SELAX_U }; // 軸ごとのビット表
        for (BYTE axis = 0; axis < 4; axis++) {
                // 軸ごとにエラー、イベント確認
                if (msts[axis] == BIT_MSTS_SERR) { // エラー発生
                        CPDLib_ReadRest(hDeviceID, PCL_NUM_1, axis_bit[axis], &ests[axis]); // エラーステータス
                }
                if (msts[axis] == BIT_MSTS_SINT) { // イベント発生
                        CPDLib_ReadRist(hDeviceID, PCL_NUM_1, axis_bit[axis], &ists[axis]); // イベントステータ
ス
                }

                LONG ret;
                if (ests[axis]) {       // エラー発生
                        ests[axis] = 0; // クリア
                        // TODO: エラー処理
                }
                else {
```

```
                        if (ists[axis]) {       // イベントステータスが立った
                                if (ists[axis] == BIT_RIST_ISEN) {      // 正常停止？
                                        ists[axis] = 0; // クリア
                                        // TODO: 正常停止処理
                                }
                                else { // その他のなんらかのイベントが発生している
                                        ists[axis] = 0; // クリア
                                        // TODO: 処理
                                }
                        }
                        else { // 動作中
                                // TODO: 動作中処理
                        }
                }
        }
```

このフローでは、イベントは正常停止時のみ監視していますが、他にイベントを監視したい場合はイベント報告設定レジスタの設定をしておきます。

1.7 動作制御

原点復帰完了後、センサ類の入出力が正しい状態になっていることを確認して動作を実行します。
動作制御ごとの手順と Windows ライブラリ関数におけるコード例を説明します。

1.7.1 位置決め

1. 動作速度の設定をします。
2. 移動量の設定をします。
3. スタートコマンドタイプを指定してスタートします。

▼リスト 1.10　位置決め動作_コード例

```
        // 速度設定
        CPD_SPEED CpdSpd;
        CpdSpd.prmg = 299;
        CpdSpd.prfl = 1000;
        CpdSpd.prfh = 5000;
        CpdSpd.prur = 1364;
        CpdSpd.prdr = 0;
        CpdSpd.prus = 0;
        CpdSpd.prds = 0;
        CpdSpd.prdp = 0;
        CpdSpd.dptype = 0;

        // 全軸に設定する
        CPDLib_SetSpeed(hDeviceID, PCL_NUM_1, SELAX_ALL, &CpdSpd);

        // 目標位置
        CPDLib_SetPosition(hDeviceID, PCL_NUM_1, SELAX_ALL, 50000);

        // 全軸加速スタート
        CPDLib_MoveRelativePosition(hDeviceID, PCL_NUM_1, SELAX_ALL, EXECMD_STAUD);
```

1.7.2 連続送り

1. 動作速度の設定をします。
2. スタートコマンドタイプと方向を指定して、スタートします。

▼リスト 1.11　連続送り_コード例

```
        // 速度設定
        CPD_SPEED CpdSpd;
        CpdSpd.prmg = 299;
        CpdSpd.prfl = 1000;
        CpdSpd.prfh = 5000;
```

```
CpdSpd.prur = 1364;
CpdSpd.prdr = 0;
CpdSpd.prus = 0;
CpdSpd.prds = 0;
CpdSpd.prdp = 0;
CpdSpd.dptype = 0;

// 全軸に設定する
CPDLib_SetSpeed(hDeviceID, PCL_NUM_1, SELAX_ALL, &CpdSpd);

// 全軸正方向、加速スタート
CPDLib_MoveContinuous(hDeviceID, PCL_NUM_1, SELAX_ALL, EXECMD_STAUD, DIRECTION_POSITIVE);
```

1.7.3 停止

1. 動作モードの設定をします。
2. スタートコマンドを書き込みます。

▼リスト 1.12　停止_コード例

```
// 全軸減速停止
CPDLib_DecStop(hDeviceID, PCL_NUM_1, SELAX_ALL);

// または即停止
CPDLib_QuickStop(hDeviceID, PCL_NUM_1, SELAX_ALL);
```

1.7.4 次動作連続実行

1. 動作用プリレジスタが空か確認 (MSTS.b14) します。
2. 空であれば、動作速度の設定、動作モードの設定をしてスタートコマンドを書き込みます。
3. 1〜2 を繰り返します。
4. 全て書き込んだ後は、動作完了を監視します。

▼リスト 1.13　次動作連続実行パラメータテーブル_コード例

```
// 次動作連続位置決め パラメータテーブル(FL速度、FH速度、移動量、動作モード)
LONG g_ContPrmTb[8][4] = {
        {1000, 10000, 20000,  EXECMD_STAFH},
        {1000, 15000, -15000, EXECMD_STAUD},
        {4000, 5000,  10000,  EXECMD_STAFL},
        {1000, 10000, 30000,  EXECMD_STAD},
        {1000, 10000, 20000,  EXECMD_STAFH},
        {1000, 15000, -15000, EXECMD_STAUD},
        {4000, 5000,  10000,  EXECMD_STAFL},
        {1000, 10000, 30000,  EXECMD_STAD},
};
```

▼リスト 1.14　次動作連続実行_コード例

```
// X軸で実行中
BYTE TbNum = 0;
while (1) {
        if (!(msts[0] & BIT_MSTS_SPRF)) {     // bit14 動作用プリレジスタが空？
                // 速度設定
                CPD_SPEED CpdSpd;
                CpdSpd.prfl = (WORD)g_ContPrmTb[TbNum][0];
                CpdSpd.prfh = (WORD)g_ContPrmTb[TbNum][1];
                CPDLib_SetSpeed(hDeviceID, PCL_NUM_1, 0x01, &CpdSpd);
                // 目標位置
                CPDLib_SetPosition(hDeviceID, PCL_NUM_1, 0x01, g_ContPrmTb[TbNum][2]);
                // スタート
                CPDLib_MoveRelativePosition(hDeviceID, PCL_NUM_1, 0x01, (BYTE)g_ContPrmTb[TbNum][3]);
                TbNum++;
        }
        if (TbNum == 7) {
```

```
                        break;
                }
        }
```

▼リスト 1.15　次動作連続実行完了監視_コード例

```
        // X軸で実行中の場合
        LONG ret;
        if (ests[0]) { // エラー発生
                ests[0] = 0; // クリア
                // TODO: エラー処理
        }
        else {
                if (!(msts[0] & BIT_MSTS_SSCM) && (msts[0] & BIT_MSTS_SEND) {      // 終了？
                        if (ists == BIT_RIST_ISEN) {   // 正常停止？
                                // TODO: 正常停止処理
                                ists[0] = 0; // クリア
                        }
                }
                else {
                        // TODO: 動作中処理
                }
        }
```

1.8　補間制御

単軸制御同様に原点復帰完了後、センサ類の入出力が正しい状態になっていることを確認して動作を実行します。

補間制御ごとの手順と Windows ライブラリ関数におけるコード例を説明します。コード例では、X と Y 軸で補間動作します。このとき、代表補間軸は X 軸です。

1.8.1　直線補間

1. 補間代表軸のみに動作速度の設定をします。
2. 補間軸全てに移動量の設定をします。
3. 補間軸全てに動作モードの設定をします。
4. 補間軸全てに合成速度一定制御の設定をします。
5. 補間軸全てにスタートコマンドタイプを指定してスタートします。

▼リスト 1.16　直線補間_コード例

```
        // 速度設定
        CPD_SPEED CpdSpd;
        CpdSpd.prmg = 299;
        CpdSpd.prfl = 1000;
        CpdSpd.prfh = 5000;
        CpdSpd.prur = 1364;
        CpdSpd.prdr = 0;
        CpdSpd.prus = 0;
        CpdSpd.prds = 0;
        CpdSpd.prdp = 0;
        CpdSpd.dptype = 0;

        // 代表補間軸のみ設定
        CPDLib_SetSpeed(hDeviceID, PCL_NUM_1, 0x01, &CpdSpd);

        // 目標位置
        // 補間する軸全てに設定
        CPDLib_SetPosition(hDeviceID, PCL_NUM_1, 0x01 | 0x02, 50000);

        // 合成速度一定制御
        // 補間する軸全てに設定
        CPDLib_SetCombinedSpeed(hDeviceID, PCL_NUM_1, 0x01 | 0x02, TRUE);

        // 加速スタート
        // 補間する軸全てに書き込み
        CPDLib_MoveLinearPosition(hDeviceID, PCL_NUM_1, 0x01 | 0x02, EXECMD_STAUD);
```

1.8.2 円弧補間

1. 補間代表軸のみに動作速度の設定をします。
2. 補間軸全てに終点位置・中心位置の設定をします。
3. 補間軸全てに円弧自動終点引き込みの設定をします。
4. 補間軸全てに動作モードの設定をします。
5. 補間軸全てに合成速度一定制御の設定をします。
6. 補間軸全てにスタートコマンドタイプを指定してスタートします。

▼リスト 1.17　円弧補間_コード例

```
// 速度設定
CPD_SPEED CpdSpd;
CpdSpd.prmg = 299;
CpdSpd.prfl = 3000;
CpdSpd.prfh = 8000;
CpdSpd.prur = 1364;
CpdSpd.prdr = 0;
CpdSpd.prus = 0;
CpdSpd.prds = 0;
CpdSpd.prdp = 0;
CpdSpd.dptype = 0;

// 代表補間軸のみ設定
CPDLib_SetSpeed(hDeviceID, PCL_NUM_1, 0x01, &CpdSpd);

// 終点・中心位置
CPD_CIRCLE CpdCir;
CpdCir.center1 = 5000;
CpdCir.center2 = 5000;
CpdCir.end1 = 5000;
CpdCir.end2 = 0;
CpdCir.retract = TRUE;
// 補間する軸それぞれに設定
CPDLib_SetCurcular(hDeviceID, PCL_NUM_1, 0x01, 0x02, &CpdCir);

// 合成速度一定制御
// 補間する軸全てに設定
CPDLib_SetCombinedSpeed(hDeviceID, PCL_NUM_1, 0x01 | 0x02, TRUE);

// スタート
// 補間する軸全てに書き込み
CPDLib_MoveCircular(hDeviceID, PCL_NUM_1, 0x01 | 0x02, EXECMD_STAFH, DIRECTION_POSITIVE);
```

1.9 終了処理

ステータスを見て軸が停止しているか確認した後、サーボドライバの場合はサーボオフを実行します。

OS ごとにデバイスリソースを破棄してボードアクセスを終了し、アプリケーションを閉じます。

Windows ライブラリ関数を用いた場合、オープンしているデバイスハンドルを指定してクローズ関数をコールしたあと、ライブラリ初期化で指定したプロダクト ID を引数としてライブラリ終了関数を実行し、ライブラリを終了してからアプリケーションを終了します。

▼リスト 1.18　デバイスクローズ_コード例

```
// ボードクローズ
CPDLib_Close(hDeviceID);

// ライブラリ終了処理
HVT_CloseLib(PRODUCT_HPCI_CPD534);
```

第 2 章

レジスタ・ポートアクセス

CPD の環境設定レジスタなど、レジスタの詳細な設定を変更したい場合は、レジスタ制御コマンドを使用してレジスタへのアクセスを行います。制御コマンドを実行するためにはポートエリアにアクセスする必要があります。ポートアクセスの詳細は、ユーザーズマニュアルをご参照ください。

オプションポートへはポートエリアの各オプションのオフセットを指定して読み書きを行います。

ここでは、Windows ライブラリ関数を用いたレジスタ・ポートアクセス方法を説明します。

2.1　レジスタアクセス

CPD のレジスタにアクセスするための関数が用意されています。

レジスタ読み出し関数をコールすると、関数内部でレジスタ読み込みコマンドを発行し、ポートの各軸入出力バッファに書き出されたデータを読み込みます。

▼リスト 2.1　レジスタ読み込み_コード例

```
// 全軸のRENV1読み込み
ULONG rreg[4];
CPDLib_ReadReg(hDevice, PCL_NUM_1, SELAX_ALL, REG_R_RENV1, &rreg[0]);
```

レジスタ書き込み関数をコールすると、関数内部でポートの各軸入出力バッファにデータを書き込み、レジスタ書き込みコマンドを発行します。

▼リスト 2.2　レジスタ書き込み_コード例

```
// X軸のRENV1書き込み
ULONG rreg = 0x20434004;
CPDLib_WriteReg(hDevice, PCL_NUM_1, 0x01, REG_R_RENV1, rreg);
```

マルチスレッドでレジスタアクセスするときは必ず排他してください。レジスタアクセスのタイミングによっては、想定外の動作をしたり不具合が発生します。詳細はユーザーズマニュアルをご参照ください。

2.2　ポートアクセス

CPD のポートにアクセスするための関数が用意されています。

メインステータス・サブステータス読み込み関数は、「1.5 ステータス取得」をご参照ください。

入出力バッファ読み書き関数は、コマンド書き込み関数と組み合わせて実行します。前章のレジスタアクセス関数を用いる場合は、これらの関数を使用する必要はありません。

▼リスト 2.3　入出力バッファ書き込み_コード例

```
// RENV1 共用体
CPD_RENV1 wreg;

// データセット
wreg.value = 0x20434004;
wreg.bf.ELM = (ULONG)1;

// X軸の出力バッファ書き込み
CPDLib_WriteBuf(hDevice, PCL_NUM_1, 0x01, wreg.value);

// X軸のRENV1書込みコマンド発行
CPDLib_WriteCmd(hDevice, PCL_NUM_1, 0x01, REG_W_RENV1);
```

▼リスト 2.4　入出力バッファ読み込み_コード例

```
// RENV1 共用体
CPD_RENV1 rreg;

// X軸のRENV1読み込みコマンド発行
CPDLib_WriteCmd(hDevice, PCL_NUM_1, 0x01, REG_R_RENV1);

// X軸の入力バッファ読み込み
CPDLib_ReadBuf(hDevice, PCL_NUM_1, 0x01, &wreg.value);
```

オプションポートアクセス関数で、オプションポートへの読み書きができます。ポート表から各ポートのオフセットを指定してコールします。

▼リスト 2.5　オプションポートアクセス_コード例

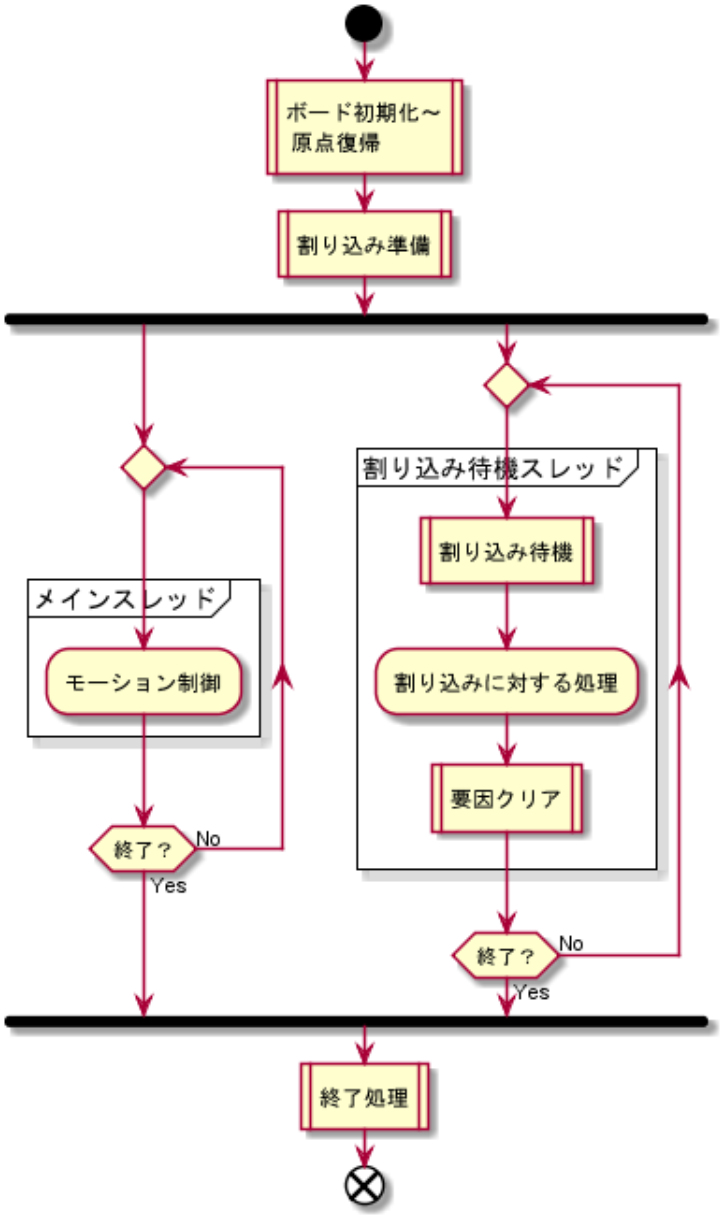
```
// ボードID読み込み
WORD bid;
CPDLib_ReadOptPortWord(hDevice, 0x9c, &bid);

// DLS/PCS入力選択
CPDLib_WriteOptPortWord(hDevice, 0x82, 1); // DLS
```

第 3 章

割り込み制御フロー解説

以下の図は、スレッドを作成して割り込み待機する制御フロー図です。



▲図 3.1　割り込み制御フロー

3.1　ボード初期化〜原点復帰

ボード初期化〜原点復帰までは第 1 章「基本制御フロー解説」と同様です。
原点復帰後、割り込みの準備をしてスレッドを生成し、割り込み待機します。

3.2　割り込み準備

ユーザーズマニュアルの「ステータスレジスタと割り込み」の割り込み設定手順を参照して割り込み準備をします。
Windows ライブラリ関数を用いた場合、軸を指定して割り込み準備関数をコールします。

▼リスト 3.1　割り込み準備_コード例

```
        // 1軸目のみ割り込み発生するように設定
        CPDLib_SetInterrupt(hDeviceID, PCL_NUM_1, 0x01);

        //スレッド起動
        DWORD dwThreadId;
        g_hThread = CreateThread(NULL, 0, WaitThreadFunc, &hDeviceID, 0, &dwThreadId);
        Sleep(100);
```

3.3　割り込み待機

ユーザーズマニュアルの「ステータスレジスタと割り込み」の割り込み要因読み出し手順を参照して割り込み待機します。

Windows ライブラリ関数を用いた場合、割り込み待機関数をコールすると内部でボード割込み許可設定を有効化してタイムアウト時間だけ割り込み待機します。割り込み待機関数の戻り値を確認して割り込みに対する処理を記述してください。割り込み発生後、次に割り込み待機するまでに割り込み要因のクリアを必ずしてください。

▼リスト 3.2　割り込み待機_コード例

```
//スレッド関数
DWORD WINAPI WaitThreadFunc(LPVOID arg)
{
        ULONG intrCount = 0;
        ULONG ret = 0;
        while (1) {
                ret = CPDLib_WaitNextInterrupt(*(ULONG*)arg, 0);
                if (ret == HVT_ERR_WAITING_CANCELED)
                        break;
                // 以下、割り込み処理
                // ・・・処理続く
        }
        return 0;
}
```

マルチスレッドで割込み以外のライブラリ関数を使用するときは必ず排他してください。
マルチスレッドで Windows ライブラリ関数以外を使ってボードにアクセスするときは、必ず排他して、レジスタへ読み書きの順序に気を付けてください。詳細はユーザーズマニュアルの「レジスタアクセスのタイミング」をご参照ください。

※注意. パソコンの性能や使用状況により割り込みを取得できない場合があります。特に OS が Windows の場合は保証できません。
また、環境に合わない極端に短い周期設定 (例えば WindowsOS で 1 ミリ秒以下など) を行った場合、イベント割込みが取得できずタイムアウトすることもあります。

3.4 終了処理

終了処理は第 1 章「基本制御フロー解説」と同様です。

CPD シリーズ　チュートリアル

2023 年 12 月 26 日　新規作成 v1.0.0

発行所　株式会社ハイバーテック
連絡先　株式会社 ハイバ－テック、東京都江東区新大橋 1-8-11 大樹生命新大橋ビル、TEL 03-3846-3801、FAX 03-3846-3773、sales@hivertec.co.jp

(C) Hivertec, Inc.