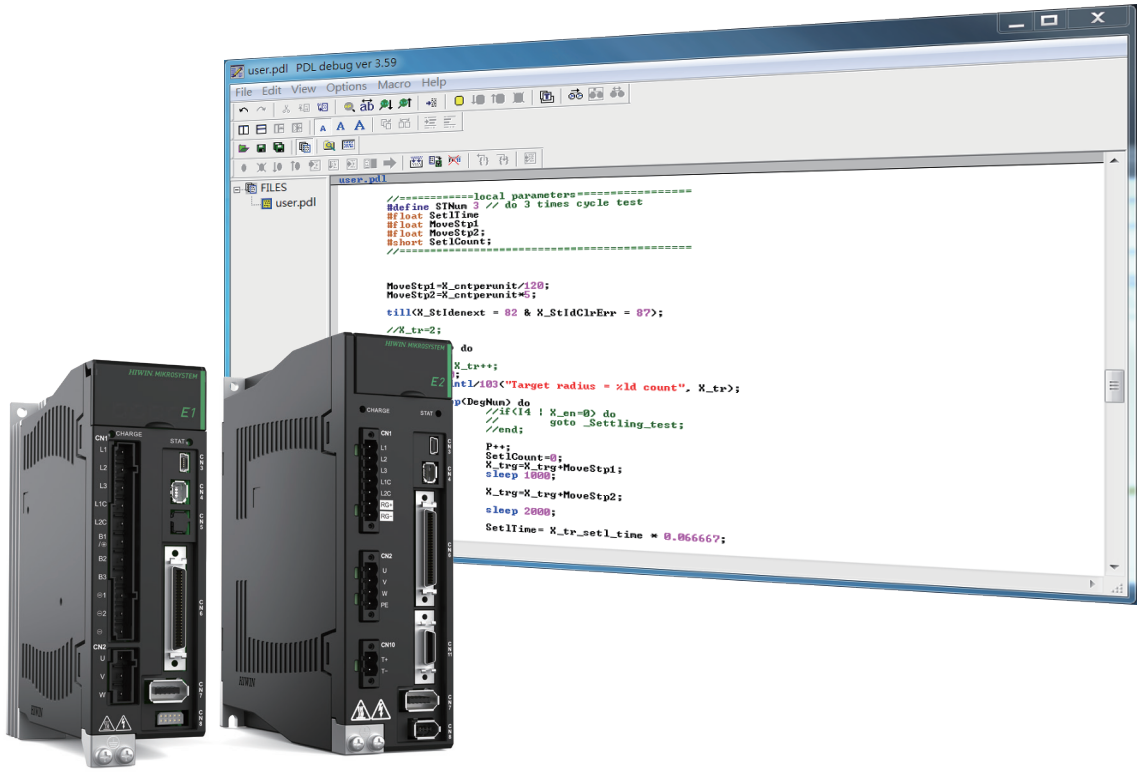
HIWIN MIKROSYSTEM



E シリーズドライバ

PDL 集

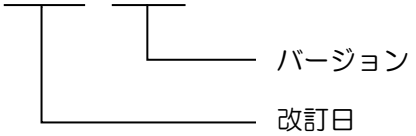
www.hiwinmikro.tw

MD25UJ01-2407_V2.2

改訂履歴

マニュアルのバージョンは表紙の下部にも記載されています。

MD25UJ01-2407_V2.2



日付	バージョン	適用機種	改訂内容
2024 年 7 月	2.2	E1 シリーズドライバー E2 シリーズドライバー	セクション 2.3 物理量モニタリング変数 (read) を更新
2023 年 4 月 25 日	2.1	E1 シリーズドライバー E2 シリーズドライバー	1. マニュアルの名前を更新 2. ドライバーモデルとソフトウェアおよびファームウェアの要件を更新
2021 年 6 月 6 日	2.0	E1 シリーズドライバー	1. 第 2 章の変数を追加 2. 追加された変数に基づいて PDL を最適化し、アプリケーションの説明を変更
2020 年 12 月 2 日	1.1	E1 シリーズドライバー	PDL を最適化し、アプリケーションの説明を変更
2020 年 2 月 20 日	1.0	E1 シリーズドライバー	初版

目次

1. サンプルプログラム集

この章では、E シリーズドライバーの PDL サンプルの応用例 (ホーミング、相対移動、ジョグ、単位変換など) をユーザーに提供します。関連する制御、物理量、サーボ変数については、第 2 章「E1 シリーズドライバーユーザーマニュアル」および「E2 シリーズドライバーユーザーマニュアル」を参照してください。

1.1 インデックス直接検索によるホーミング

1.1.1 基本情報

サンプル番号	1
ファイル名	1.pdl
ドライバーモデル	E シリーズドライバー
モータータイプ	AC サーボモーター, リニアモーター, ダイレクトドライブモーター, トルクモーター
ソフトウェアおよびファームウェアの要件	E1 シリーズドライバー: Thunder 1.6.$.$ MDP 2.6.$ 以上 E2 シリーズドライバー: Thunder 1.9.$.$ MDP 3.9.$ 以上
適用モード	内部位置モード
機能	デジタル入力 I2 をトリガーして、ドライバーの組み込みホーミング機能を実行します。ホーミングが完了するとすぐに、デジタル出力 O2 がホストコントローラーにパルスを出力します。
使用する入力	I2
使用する出力	O2

注: このサンプルプログラムを使用する前に、Thunder のメインウィンドウに移動し、メニューバーで [Tools] を選択して [IO configuration] をクリックし、バッティングを避けるために I2 と O2 を [Not configure] に設定してください。

1.1.2 サンプルプログラム

```
// ============================ Macro ============================
//   Users can set homing velocity and output pulse width in this macro.
#define SEARCH_DIR   34                  //   Search index in positive direction
//   #define SEARCH_DIR   33             //   Search index in negative direction
#define R_SPEED_SEARCH_FAST   30         //   rotary velocity for finding near home sensor
                                         //   (Unit: rpm)
#define R_SPEED_SEARCH_SLOW   10         //   rotary velocity for finding home position (Unit: rpm)
#define L_SPEED_SEARCH_FAST   20         //   linear velocity for finding near home sensor
                                         //   (Unit: mm/s)
```

```
#define L_SPEED_SEARCH_SLOW    10          //  linear velocity for finding home position (Unit: mm/s)
#define HOMEING_TIMER    30                //  time limit for homing procedure (Unit: s)
#define OUTPUT_WIDTH    2000               //  output pulse width (Unit: ms)
proc pdl_en() do                           //  Enable the motor
      mEnable = 1;
      till (S_RDY);
end;
proc pdl_dis() do                          //  Disable the motor
      mEnable = 0;
      till (~S_RDY);
end;
proc home_para() do                        //  Set homing parameters
      Pt700 = SEARCH_DIR;                  //  homing method
      Pt701 = R_SPEED_SEARCH_FAST;
      Pt702 = R_SPEED_SEARCH_SLOW;
      Pt703 = HOMEING_TIMER;
      Pt705 = L_SPEED_SEARCH_FAST;
      Pt706 = L_SPEED_SEARCH_SLOW;
end;
proc homing() do
      home_para();
      pdl_en();
      till (S_RDY);
      mHome = 1;                           //  Call servo drive's built-in homing command
      sleep 100;                           //  Wait for 100 ms
      till (dHomeState = 3);               //  Check homing procedure (in the process)
      till (dHomeState = 2 | dHomeState = -1);   //  Check homing procedure (succeed or fail)
      if (dHomeState <> 2) do              //  Check if homing succeeds
            printl/101("Homing process failed.");
            pdl_dis();
      end;
      sleep 1000;
end;
// ======================== Main Program ========================
#task/1;                                   //  Create task #1
till (~I2);                                //  Wait for I2 to be OFF
setoff O2;                                 //  Set O2 to be OFF
// ======================== Homing Procedure ========================
till (I2);                                 //  Wait for I2 to be ON
```

```
homing();                                    //  Call homing procedure
// After homing succeeds, the servo drive outputs a pulse to the host controller.
If (dHomeState = 2) do                       //  Check if homing succeeds
    seton O2;                                //  Set O2 to be ON
    sleep OUTPUT_WIDTH;                      //  Wait for 2 s
    setoff O2;                               //  Set O2 to be OFF
    sleep 1000;                              //  Wait for 1 s
    pdl_dis();
end;
ret;
```

1.1.3 使い方

このサンプルコードは、デジタル入力 I2 をトリガーして組み込みのホーミング機能を実行します。ホーミング手順が終了すると、デジタル出力 O2 がホストコントローラーにパルスを出力します。このサンプルコードのフローチャートを図 1.1.3.1 に示します。

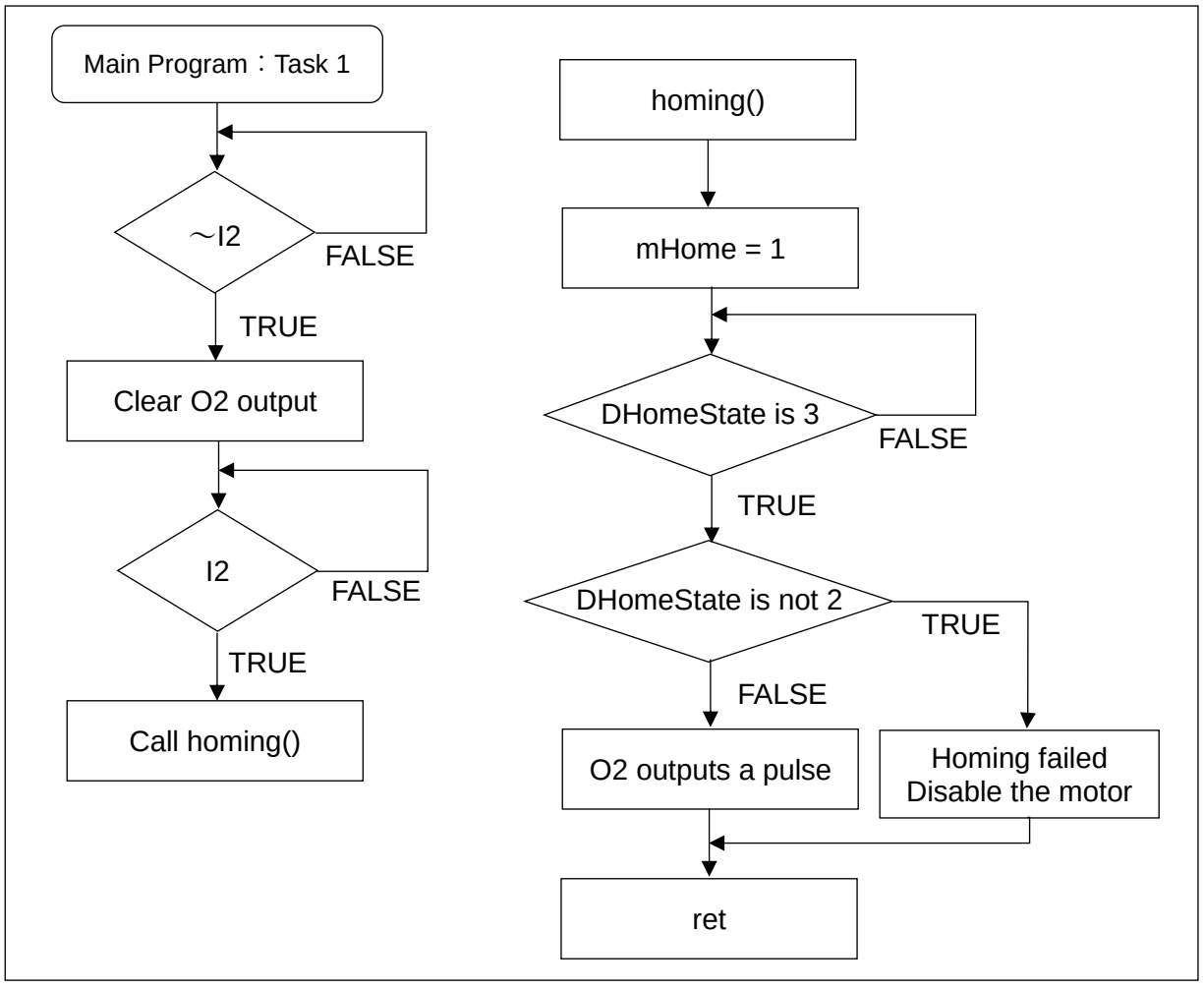


図 1.1.3.1

1.1.4 アプリケーションの説明

メインプログラムの先頭で、#task/1 と呼ばれるタスクが開始されます。これは、ドライバーのマルチタスクプロセスで、ユーザーが編集した PDL アプリケーションプログラムにタスク番号を割り当てるためのものです。E シリーズドライバーは、ユーザーが自由に PDL を編集できるようにタスク番号 1 ～ 3 を提供します。

ドライバーのオペレーティングシステム（OS）には、mHome というコマンドがあります。これは、ドライバーに組み込まれているホーミング用のコマンドです。プログラミングコードで mHome＝1 を使用すると、ホーミングメソッド 34 でインデックス信号を直接検索してホーミングが実行されます。ホーミング用の変数は、図 1.1.4.1 に示すように、Thunder の「Homing Operation」ウィンドウで設定できます。

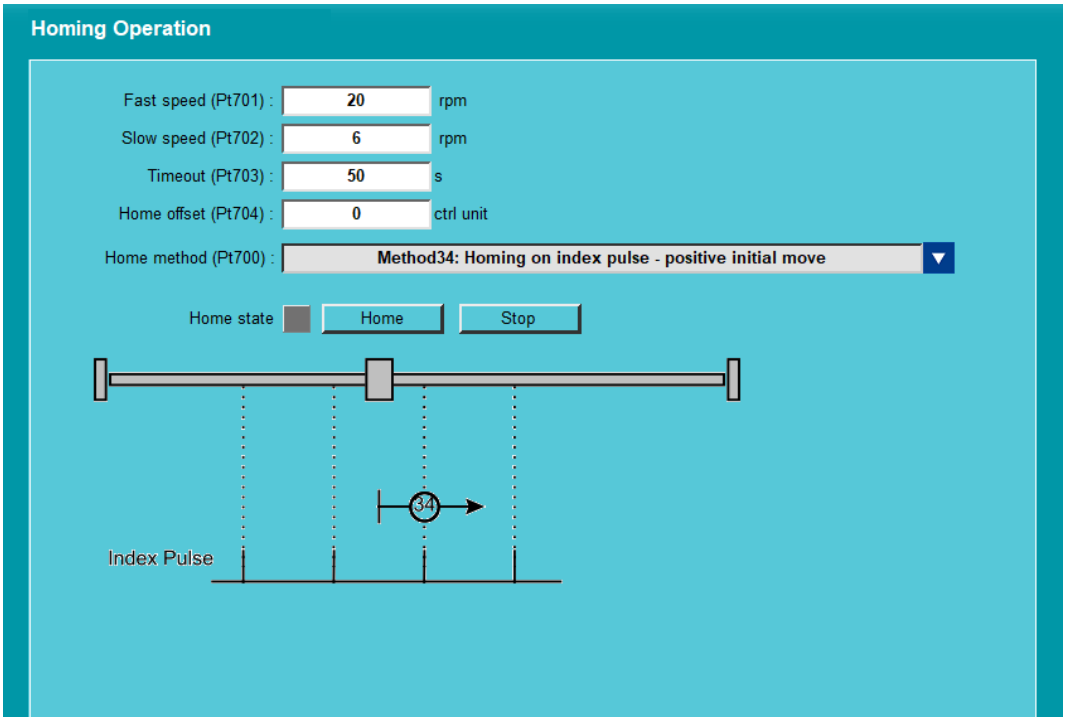


図 1.1.4.1 ホーミング変数の設定

このサンプル コードには、表 1.1.4.1 に示すように、ホーミング状態を表示するための dHomeState という変数があります。

表 1.1.4.1 dHomeState の定義

dHomeState	ホーミング状態
-1	Homing fails
0	Homing not executed
3	Homing
4	Homing stops
2	Homing succeeds

1.2 ハードストップタッチによるホーミング

1.2.1 基本情報

サンプル番号	2
ファイル名	2.pdl
ドライバーモデル	E シリーズドライバー
モータータイプ	AC サーボモーター（ボールねじ付き）、リニアモーター、ダイレクトドライブモーター、トルクモーター
ソフトウェアおよびファームウェアの要件	E1 シリーズドライバー: Thunder 1.6.$.$ MDP 2.6.$ 以上 E2 シリーズドライバー: Thunder 1.9.$.$ MDP 3.9.$ 以上
適用モード	内部位置モード
機能	モーターはまず負方向へ移動し、ハードストップに接触して出力電流が予め定義されたしきい値に達します。その後、モーターは向きを変えて正方向へ移動し、インデックスを検索してホーミングを終了します。
使用する入力	I2
使用する出力	なし

注: このサンプルプログラムを使用する前に、Thunder のメインウィンドウに移動し、メニューバーで [Tools] を選択して [IO configuration] をクリックし、バッティングを避けるために I2 を [Not configure] に設定してください。

1.2.2 サンプルプログラム

```
// ============================ Macro ============================
//   Users can decide the start direction of homing, and set velocity parameters based on motor type.
#define SEARCH_DIR   1                    //   Start homing in negative direction
//   #define SEARCH_DIR   2                //   Start homing in positive direction
#define BackDistance   10000              //   distance for leaving hard stop (Unit: control unit)
#define P_Constant_Vel   1                //   Positive constant velocity motion
#define N_Constant_Vel   -1               //   Negative constant velocity motion
#define R_SPEED_SEARCH_FAST   30          //   rotary velocity for finding near home sensor
                                          //   (Unit: rpm)
#define R_SPEED_SEARCH_SLOW   10          //   rotary velocity for finding home position (Unit: rpm)
#define L_SPEED_SEARCH_FAST   20          //   linear velocity for finding near home sensor
                                          //   (Unit: mm/s)
#define L_SPEED_SEARCH_SLOW   10          //   linear velocity for finding home position (Unit: mm/s)
#define HOMEING_TIMER   30                //   time limit for homing procedure (Unit: s)
#define CONTINUOUS_CURRENT   3.5          //   continuous current (Unit: A-amp)
#long Start_dir;
```

```
proc pdl_en() do                                    //  Enable the motor
    mEnable = 1;
    till (S_RDY);
end;
proc pdl_dis() do                                   //  Disable the motor
    mEnable = 0;
    till (~S_RDY);
end;
proc home_para() do                                 //  Set homing parameters
    if (Start_dir = 1) do
        Pt700 = 34;                                 //  homing method
    else do
        Pt700 = 33;                                 //  homing method
    end;
    Pt701 = R_SPEED_SEARCH_FAST;
    Pt702 = R_SPEED_SEARCH_SLOW;
    Pt703 = HOMEING_TIMER;
    Pt705 = L_SPEED_SEARCH_FAST;
    Pt706 = L_SPEED_SEARCH_SLOW;
end;
proc homing() do
    home_para();
    pdl_en();
    till (S_RDY);
    mHome = 1;                                      //  Call servo drive's built-in homing command
    sleep 100;                                      //  Wait for 100 ms
    till (dHomeState = 3);                          //  Check homing procedure (in the process)
    till (dHomeState = 2 | dHomeState = -1);        //  Check homing procedure (succeed or fail)
    if (dHomeState <> 2) do                         //  Check if homing succeeds
        printl/101("Homing process failed.");
        pdl_dis();
    end;
    sleep 1000;
end;
// ======================== Main Program ========================
#task/1;                                            //  Create task #1
#float CurrThreshold;
mTestRun = 1;                                       //  Forcibly switch to internal position test run mode
Start_dir = SEARCH_DIR;
```

```
CurrThreshold = CONTINUOUS_CURRENT*1.5;  //  Set the threshold for output current (Unit: A-amp)
CurrThreshold*= CurrThreshold;
Pt533 = 50;                              //  rotary velocity for searching hard stop (Unit: rpm)
Pt585 = 50;                              //  linear velocity for searching hard stop (Unit: mm/s)
Pt534 = 100;                             //  soft start acceleration time (Unit: ms)
Pt537 = 100;                             //  soft start deceleration time (Unit: ms)
till (~I2);                              //  Wait for I2 to be OFF
till (I2);                               //  Wait for I2 to be ON
pdl_en();
till (S_RDY);
printl/103("Start to search hard stop.");
if (Start_dir = 1) do
    mRun = P_Constant_Vel;               //  Touch hard stop in negative direction
else do
    mRun = N_Constant_Vel;               //  Touch hard stop in positive direction
end;
till (dFbCurr > CurrThreshold | ALM);    //  Wait until the current threshold is reached
                                         //  dFbCurr is the motor current (Unit: A-amp^2)
mStopMove = 1;                           //  Motion stop
if (ALM) do
    printl/101("Search hard stop failed!");
    goto _end;
end;
till (~X_run);
printl/103("Hard stop found, going back.");
sleep 100;
if (Start_dir = 1) do
    mMovePos += BackDistance;            //  Leave hard stop
else do
    mMovePos -= BackDistance;            //  Leave hard stop
end;
sleep 100;
till (~X_run);
printl/103("Start to search index.");
homing();
printl/103("Homing process succeeded.");
_end:
pdl_dis();
ret;
```

1.2.3 使い方

ホーミングプロセスでは、リミットスイッチまたはハードストップに触れることを選択できます。これにより、インデックス検索が毎回同じ参照ポイントから同じ方向で開始されることが保証されます。リミットスイッチがない場合は、参照ポイントとしてハードストップに触れる方法を選択できます。

ハードストップにタッチする方法は、同じ側でハードストップにタッチし、次に逆方向に進んでインデックス信号を検索することです。これにより、良好なホーム再現性が得られます。ハードストップにタッチするホーム復帰方法のフローチャートを図 1.2.3.1 に示します。

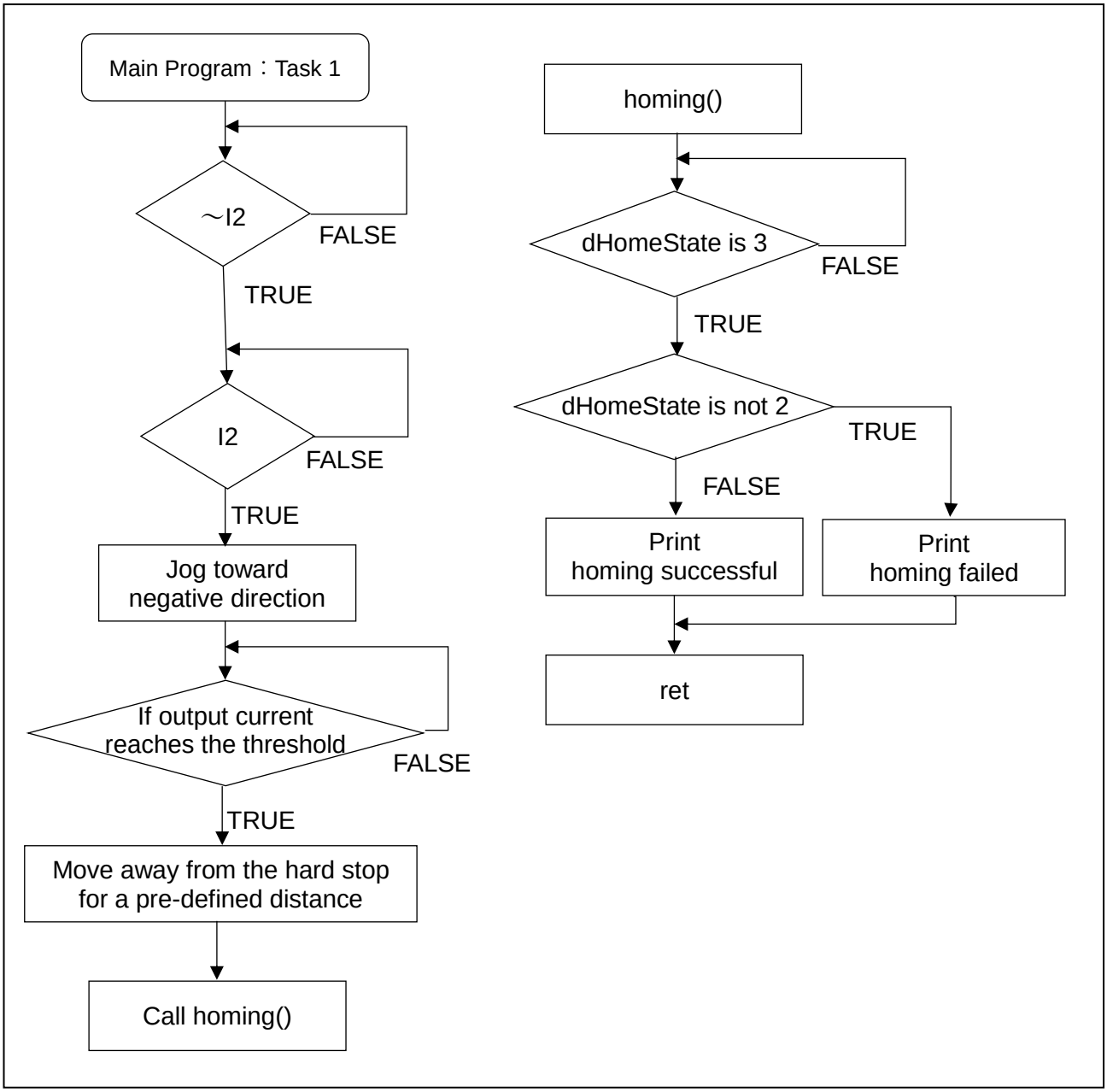


図 1.2.3.1

最初に、連続電流 (CONTINUOUS_CURRENT) と出力電流のしきい値 (CurrThreshold) を設定する必要があります。設定プロセスは次のとおりです。

Step 1. Thunder の「Configuration Wizard」ウィンドウを開きます。
Step 2. 「Motor Setup」タブに移動して、「Continuous current」の値を確認します (単位: A-rms)。
Step 3. 単位 A-rms を A-amp (式: A-rms x 1.414 = A-amp) に変換し、その値を CONTINUOUS_CURRENT として設定します。
Step 4. CurrThreshold を CONTINUOUS_CURRENT の値より少し大きく設定します。推奨値は 1.5 倍です。

注意: 電流しきい値をあまり大きく設定しないでください。大きく設定すると、モーターが急停止したときに「Overcurrent detected」というアラームが発生する可能性があります。

1.2.4 アプリケーションの説明

リミットスイッチがトリガーされた後にインデックスを検索する場合は、Thunder の「Homing Operation」ウィンドウに移動して「Method1」を選択します。I

1.3 I/O を使用して移動をトリガーする

1.3.1 基本情報

サンプル番号	3
ファイル名	3.pdl
ドライバーモデル	E シリーズドライバー
モータータイプ	AC サーボモーター、リニアモーター、ダイレクトドライブモーター、トルクモーター
ソフトウェアおよびファームウェアの要件	E1 シリーズドライバー: Thunder 1.6.$.$　MDP 2.6.$ 以上 E2 シリーズドライバー: Thunder 1.9.$.$　MDP 3.9.$ 以上
適用モード	内部位置モード
機能	デジタル入力 I2 と I3 をトリガーして、モーターを事前に定義された距離だけ移動させます。I2 は正の動き、I3 は負の動きを表します。
使用する入力	I2, I3
使用する出力	なし

注意：

(1) リニアモーターを使用する場合は移動距離の制限に注意してください。

(2) このサンプルプログラムを使用する前に、Thunder のメインウィンドウに移動し、メニューバーで[Tools]を選択し、[IO configuration]をクリックして、バッティングを避けるために I2 と I3 を [Not configure]に設定します。

1.3.2 サンプルプログラム

```
// ============================ Macro ============================
//   The moving velocity of motor is determined by the setting of Pt533 or Pt585.
#define Distance   1000                          //   Set moving distance (Unit: control unit)
proc pdl_en() do                                 //   Enable the motor
      mEnable = 1;
      till (S_RDY);
end;
proc pdl_dis() do                                //   Disable the motor
      mEnable = 0;
      till (~S_RDY);
end;
// ========================= Main Program =========================
#long io_switch, io;
#task/1;                                         //   Create task #1
```

```
mTestRun = 1;                          //  Forcibly switch to internal position test run mode
io_loop:
if (I2& ~I3) do                        //  I2 is ON and I3 is OFF
      io=1;
      io_switch=1;
      call _trigger_loop;
end;
if (I3 & ~I2) do                       //  I3 is ON and I2 is OFF
      io=2;
      io_switch=1;
      call _trigger_loop;
end;
goto io_loop;
ret;

_trigger_loop:
if (io_switch=1) do
      pdl_en();
      till (S_RDY);
      if (io=1) do                     //  If I2 is ON and I3 is OFF, positive move.
            mMovePos += Distance;      //  Positive move for a pre-defined distance
            sleep 50;
            till (COIN);               //  Wait for the motor to stop
            printl/103("Positive motion ended.");
            sleep 1000;
      end;
      if (io=2) do                     //  If I3 is ON and I2 is OFF, negative move.
            mMovePos -= Distance;      //  Negative move for a pre-defined distance
            sleep 50;
            till (COIN);               //  Wait for the motor to stop
            printl/103("Negative motion ended.");
            sleep 1000;
      end;
end;
io_switch=0;
io=0;
pdl_dis();
till (~I2 & ~I3);                      //  Ensure I2 and I3 are reset to the initial status
ret;
```

1.3.3 使い方

ユーザーはまず移動距離を設定する必要があります。次に、デジタル入力 I2 または I3 をトリガーして、モーターを正方向または負方向に移動します。プログラムが _trigger_loop を実行する前に、I2 と I3 の状態がチェックされます。I2 と I3 の機能は、表 1.3.3.1 に示されています。

表 1.3.3.1 I2 と I3 の機能

I2	I3	動作方向
ON	OFF	正方向
OFF	ON	負方向

1.3.4 アプリケーションの説明

PDL では、mMovePos は絶対位置決めをトリガーするコマンドです。mMovePos の値がエンコーダーのフィードバック値と異なる場合、モーターは mMovePos の位置に移動します。また、モーターが移動しているときに、ユーザーは mMovePos の値を変更できます。この場合、モーターは mMovePos の最新の設定位置に移動します。

従って、相対的な位置決めを実現するには、目標位置を基準点として使用する必要があります。

```
mMovePos += Distance;
```

この例の最後のヒントは、printl/retprintl コマンドです。これらは、「Message + command prompt」ウィンドウにメッセージを表示するために使用されます。

1.4 単位変換

1.4.1 基本情報

サンプル番号	4
ファイル名	4.pdl
ドライバーモデル	E シリーズドライバー
モータータイプ	AC サーボモーター、リニアモーター、ダイレクトドライブモーター、トルクモーター
ソフトウェアおよびファームウェアの要件	E1 シリーズドライバー: Thunder 1.6.$.$　MDP 2.6.$ 以上 E2 シリーズドライバー: Thunder 1.9.$.$　MDP 3.9.$ 以上
適用モード	内部位置モード
機能	単位を「mm」または「度」から「制御単位」に自動的に変換します。これにより、アプリケーションエンジニアは、後のアプリケーションで単位変換に煩わされる必要がなくなります。
使用する入力	なし
使用する出力	なし

注意: ボールねじ付き AC サーボモーターの場合、ユーザーは最初に外部エンコーダー (Pt20A) の送り長さを設定する必要があります。

1.4.2 サンプルプログラム

```
// ============================ Macro ============================
#define Distance   45                          //   Set moving distance (mm or deg)
// ========================= Main Program =========================
// =   Description:  For linear motor – convert mm to control unit
// =                 For AC servo motor with ball screw – convert mm to control unit (Set Pt20A first)
// =                 For direct drive motor / torque motor – convert deg to control unit
// =   Arguments:   NULL
// ================================================================
#task/1;                                       //   Create task #1
#float LMCntPerMM;
#float DisTemp;
#long Rev2mm                                   //   Pt20A feed length of external encoder
Rev2mm = Pt20A/1000;                           //   um -> mm

if (dMotorType=2) do                           //   the motor is AC servo motor
```

```
    DisTemp = Distance;
    DisTemp /= Rev2mm;                  //  mm -> rev
    DisTemp *= dCntPerUnit;             //  rev -> count
    DisTemp *= Pt210;                   //  electronic gear ratio (denominator)
    DisTemp /= Pt20E;                   //  electronic gear ratio (numerator)
                                        //  count -> control unit
end;

if (dMotorType=1) do                    //  the motor is direct drive motor / torque motor
    DisTemp = Distance;
    DisTemp /= 360;                     //  deg -> rev
    DisTemp *= dCntPerUnit;             //  rev -> count
    DisTemp *= Pt210;                   //  electronic gear ratio (denominator)
    DisTemp /= Pt20E;                   //  electronic gear ratio (numerator)
                                        //  count -> control unit
end;

if (dMotorType=0) do                    //  the motor is linear motor
    DisTemp = Distance;
    LMCntPerMM = dCntPerUnit / 100;     //  count / 100mm -> count / mm
    DisTemp *= LMCntPerMM;              //  mm -> count
    DisTemp *= Pt210;                   //  electronic gear ratio (denominator)
    DisTemp /= Pt20E;                   //  electronic gear ratio (numerator)
                                        //  count -> control unit
end;

ret;
```

1.4.3 使い方

PDL では、すべてのモーション変数 (速度 Pt533、加速度 Pt534、減速度 Pt537 など) と位置変数 (mMovePos) は、単位として「制御単位」を使用します。したがって、入力された物理量は、まず「カウント」に変換し、次に電子ギア比を介して「制御単位」に変換する必要があります。このサンプルコードは、モーションを実行する前に単位を変換する方法を示しています。

たとえば、リニアモーターを現在の位置から 45 mm 移動する必要があるとします (このサンプルコードに示すように、#define Distance 45)。計算結果は DisTemp に保存されるため、次のコマンドでモーターを 45 mm 移動できます。

```
    mMovePos += DisTemp;
```

1.4.4 アプリケーションの説明

このサンプルコードには、Pt20A という組み込み変数があります。この変数は、AC サーボモーターのネジピッチを表します。また、dMotorType は、表 1.4.4.1 に示すように、モーターの種類を表します。

表 1.4.4.1 dMotorType の定義

dMotorType	モータータイプ
0	リニアモーター
1	ダイレクトドライブモーター / トルクモーター
2	AC サーボモーター

この例では、別の組み込み変数 dCntPerUnit があります。この変数はエンコーダーの分解能に対応します。各モーター タイプに対する dCntPerUnit の定義は、表 1.4.4.2 に示されています。

表 1.4.4.2 各モータータイプの dCntPerUnit の定義

モータータイプ	dCntPerUnit
リニアモーター	count / 100mm
ダイレクトドライブモーター / トルクモーター	count / rev
AC サーボモーター	count / rev

1.5 ティーチング機能

1.5.1 基本情報

サンプル番号	5
ファイル名	5.pdl
ドライバーモデル	E シリーズドライバー
モータータイプ	AC サーボモーター、リニアモーター、ダイレクトドライブモーター、トルクモーター
ソフトウェアおよびファームウェアの要件	E1 シリーズドライバー: Thunder 1.6.$.$ MDP 2.6.$ 以上 E2 シリーズドライバー: Thunder 1.9.$.$ MDP 3.9.$ 以上
適用モード	内部位置モード
機能	この例では、ユーザーはデジタル入力をトリガーして目的の位置をティーチングして記録できます。その後、ユーザーは対応するデジタル入力をトリガーして、モーターをティーチングポイントに移動させることができます。
使用する入力	I2, I3, I4, I6
使用する出力	なし

注: このサンプルプログラムを使用する前に、Thunder のメインウィンドウに移動し、メニューバーで [Tools] を選択して [IO configuration] をクリックし、バッティングを避けるために I2、I3、I4、および I6 を [Not configure] に設定してください。

1.5.2 サンプルプログラム

```
// ============================ Macro ============================
#define BufferSize   4                          //  Define the buffer size for teaching points
proc pdl_en() do                                //  Enable the motor
    mEnable = 1;
    till (S_RDY);
end;
proc pdl_dis() do                               //  Disable the motor
    mEnable = 0;
    till (~S_RDY);
end;
// ======================== Global Variable ========================
#short BufferNum;                               //  Position buffer index
#long PosBuff[BufferSize];                      //  Position buffer for storing each teaching point
// ========================= Main Program =========================
#task/1;                                        //  Create task #1
```

```
BufferNum=0;                                    //  Initialize the index of position buffer
mTestRun = 1;                                   //  Forcibly switch to internal position test run mode
_trigger_loop:
if (I2) do                                      //  If I2 is ON, call the process of teaching point.
    call _PosMem;                               //  Call the process to teach point
    till (~I2);
end;
if (I6) do                                      //  If I6 is ON, call the process of moving.
    pdl_en();
    till (S_RDY);
    if (mEnable=0) do                           //  If motor is disabled
        printl/101("Motor is not enabled!");    //  Print the warning message for disabled motor
        till (~I6);
        goto _trigger_loop;                     //  When I6 = 0, go back to _trigger_loop.
    end;
    call _Move2Tar;                             //  Call the process of moving to a teaching point
    till (~I6);
end;
goto _trigger_loop;
ret;

// ======================= Teaching (PosMem) =======================
// =   Description:  Save position in position registers PosBuff according to position index BufferNum
// =   Arguments:   NULL
// =================================================================
_PosMem:
#long posTemp;
posTemp=dFbPos;
PosBuff[BufferNum]= posTemp;                    //  Save feedback position to PosBuff
printl/103("Teaching point %ld, Position=%ld", BufferNum, posTemp);
//   Print the saved number and position of teaching point
BufferNum+=1;                                   //  Move to next index of PosBuff
    if (BufferNum=BufferSize) do                //  If index exceeds the buffer size
        BufferNum=0;                            //  Go back to the 1st buffer
    end;
ret;
// ================== Move to teaching point (Move2Tar) ==================
// =   Description:  Choose the target position by I4 and I3
// =   Arguments:   NULL
```

```
// ============================================================
_Move2Tar:
#short index;
if (~I4 & ~I3) do                          //   (I4, I3) = (0, 0)
      index =0;
end;
if (~I4 & I3) do                           //   (I4, I3) = (0, 1)
      index=1;
end;
if (I4 & ~I3) do                           //   (I4, I3) = (1, 0)
      index=2;
end;
if (I4 & I3) do                            //   (I4, I3) = (1, 1)
      index=3;
end;
mMovePos=PosBuff[index];                   //   Move to selected teaching point
till (~X_run);                             //   Wait for the motor to stop
printl/103("Move to point %ld ended.", index);   //   Print the in-position information
printl/103("Position=%ld", PosBuff[index]);      //   Print the position
sleep 100;
ret;
```

1.5.3 使い方

ティーチング機能はデジタル入力によって起動されます。使用される入力の機能は表 1.5.3.1 に示されています。

表 1.5.3.1 使用される入力の機能

デジタル入力	機能
I2	立ち上がりエッジトリガー。位置をティーチングします。現在の位置をバッファに記録します。
I3	ティーチングポイントを選択します。
I4	
I6	立ち上がりエッジトリガー。選択したティーチングポイントに移動します。

I2 が 1 回トリガー (立ち上がりエッジトリガー) されると、ティーチングプロセスが 1 回実行されます。表 1.5.3.2 は、I2 のトリガー時間とティーチングポイントの関係を示しています。I2 が 4 回以上トリガーされた場合、後のトリガーは位置 0 から始まる前のティーチングポイントを順番に上書きします。ティーチングプロセスを再開する場合は、ドライバーをリセットするか、電源を再起動することによってのみ実行できます。

注意：
リセットを行うと、以前のティーチングポイントはすべてクリアされます。また、すべての初期ティーチングポイントは 0 に設定されます。

表 1.5.3.2 I2 のトリガー時間とティーチングポイントの関係

Index	ティーチングポイント
0	現在の位置をティーチングポイント 0 に記録する
1	現在の位置をティーチングポイント 1 に記録する
2	現在の位置をティーチングポイント 2 に記録する
3	現在の位置をティーチングポイント 3 に記録する

ティーチングポイントへの動作を開始する前に、まずパラメーター設定で移動速度、加速度、および対応する動作変数を設定する必要があります。次に、I4 と I3 を使用してティーチングポイントをターゲット位置として選択し、I6 を使用して動作をトリガーします。表 1.5.3.3 に、デジタル入力とアクションの関係を示します。

表 1.5.3.3 デジタル入力とアクションの関係

入力			アクション
I4	I3	I6	
OFF	OFF	立ち上がりエッジトリガー	ティーチングポイント 0 へ移動
OFF	ON		ティーチングポイント 1 へ移動
ON	OFF		ティーチングポイント 2 へ移動
ON	ON		ティーチングポイント 3 へ移動

1.6 未完了の絶対位置決めを再開できるようにする

1.6.1 基本情報

サンプル番号	6
ファイル名	6.pdl
ドライバーモデル	E シリーズドライバー
モータータイプ	AC サーボモーター、リニアモーター、ダイレクトドライブモーター、トルクモーター
ソフトウェアおよびファームウェアの要件	E1 シリーズドライバー: Thunder 1.6.$.$　MDP 2.6.$ 以上 E2 シリーズドライバー: Thunder 1.9.$.$　MDP 3.9.$ 以上
適用モード	内部位置モード
機能	モーターは絶対座標で移動するため、動作中にエラーが発生して動作が停止することがあります。エラーが修復され、モーターが再び動作可能になると、動作を停止する前に設定されていた元の目標位置に移動します。
使用する入力	I2, I3
使用する出力	なし

注: このサンプルプログラムを使用する前に、Thunder のメインウィンドウに移動し、メニューバーで [Tools] を選択して [IO configuration] をクリックし、バッティングを避けるために I2 と I3 を [Not configure] に設定してください。

1.6.2 サンプルプログラム

```
// ============================ Macro ============================
#define TargetPos1    Pt531                    //   target position 1 (Unit: control unit)
#define TargetPos2    Pt532                    //   target position 2 (Unit: control unit)
proc pdl_en() do                               //   Enable the motor
      mEnable = 1;
      till (S_RDY);
end;
proc pdl_dis() do                              //   Disable the motor
      mEnable = 0;
      till (~S_RDY);
end;
// ========================= Main Program =========================
#task/1;                                       //   Create task #1
#long targetpos;
#long MotionCompleteFg;                        //   Flag is used to identify whether motor is at target or not
```

```
mTestRun = 1;                          //  Forcibly switch to internal position test run mode
MotionCompleteFg = 1;                  //  Motor arrives at the target position
_TargetMove:
till (~I2 & ~I3);
till (I2 | I3);
if (I2 & ~I3) do
    targetpos = TargetPos1;
    MotionCompleteFg = 0;              //  Motor does not arrive at the target position
end;
if (~I2 & I3) do
    targetpos = TargetPos2;
    MotionCompleteFg = 0;
end;
pdl_en();
while (MotionCompleteFg = 0) do
    till (S_RDY);                      //  If motor stops due to an error, wait here until it is re-enabled.
    mMovePos = targetpos;              //  Start to move to the target position
    sleep 100;
    till (~X_run);                     //  Wait for motion to be completed, or stopped due to an error.
    if (COIN) do
        MotionCompleteFg = 1;          //  If motor successfully arrives, leave the while loop.
    end;
end;
goto _TargetMove;
ret;
```

1.6.3 使い方

ユーザーが絶対目標位置 (TargetPos1、TargetPos2) を設定した後、I2 と I3 をトリガーして目標位置を選択できます (表 1.6.3.1 を参照)。その後、モーターは選択した位置に向かって移動します。エラーの発生により、モーターが動作中に停止して無効になった場合は、エラーが修正されるまで待機します。モーターが有効になると、mMovePos が再度実行され、目標位置に向かって移動し続けます。モーターは、目標位置に到達するまで動作を停止します。

表 1.6.3.1 デジタル入力の機能

I2	I3	変数名	目標位置
ON	OFF	TargetPos1	Pt531
OFF	ON	TargetPos2	Pt532

1.6.4 アプリケーションの説明

このサンプルコードと前のサンプルコードでは、絶対座標で動作を行うために mMovePos が使用されています。位置コマンド mMovePos の値がフィードバック位置 dFbPos の値と等しくない場合、モーターは dFbPos が mMovePos と等しくなるまで mMovePos に向かって移動します。その後、モーターは動作を停止します。

モーターが無効により停止した場合、モーターが再度有効になった後、位置コマンド mMovePos はフィードバック位置 dFbPos の現在の値に自動的に設定されます。したがって、モーターを無効にするエラーが発生した場合、通常の状態では、再度有効になったモーターは移動せずに停止します。

さらに、このサンプルコードでは、「while ループ」を設計し、モーターが目標位置に到達した状況を表す変数 MotionCompleteFg を宣言しています。モーターの動きが完了すると、プログラムはモーターが有効か無効かをチェックします。モーターが有効であれば、モーターが実際に目標位置に到達したことを意味します。モーターが無効であれば、モーターが目標位置に到達していないことを意味します。

1.7 デジタル入力によるジョグ動作

1.7.1 基本情報

サンプル番号	7
ファイル名	7.pdl
ドライバーモデル	E シリーズドライバー
モータータイプ	AC サーボモーター、リニアモーター、ダイレクトドライブモーター、トルクモーター
ソフトウェアおよびファームウェアの要件	E1 シリーズドライバー: Thunder 1.6.$.$　MDP 2.6.$ 以上 E2 シリーズドライバー: Thunder 1.9.$.$　MDP 3.9.$ 以上
適用モード	内部位置モード
機能	ドライバーの入力を通じてジョグ動作をトリガーします。
使用する入力	I2, I3
使用する出力	なし

注: このサンプルプログラムを使用する前に、Thunder のメインウィンドウに移動し、メニューバーで [Tools] を選択して [IO configuration] をクリックし、バッティングを避けるために I2 と I3 を [Not configure] に設定してください。

1.7.2 サンプルプログラム

```
// ============================ Macro ============================
#define CW_JOG    I2                    //  Define I2 as the port for positive jog
#define CCW_JOG    I3                   //  Define I3 as the port for negative jog
#define P_Constant_Vel    1             //  Positive constant velocity motion
#define N_Constant_Vel    -1            //  Negative constant velocity motion
proc pdl_en() do                        //  Enable the motor
      mEnable = 1;
      till (S_RDY);
end;
proc pdl_dis() do                       //  Disable the motor
      mEnable = 0;
      till (~S_RDY);
end;
// ========================= Main Program =========================
#task/1;                                //  Create task #1
sleep 1000;
```

```
mTestRun = 1;                                   //  Forcibly switch to internal position test run mode
pdl_en();

_loop:
if (S_RDY) do                                   //  If the motor is enabled
    if (CW_JOG) do                              //  If the input for positive jog is ON
        mRun = P_Constant_Vel;                  //  Positive jog with the value of Pt533 or Pt585
        till (~CW_JOG);                         //  Until the input for positive jog is released
        mStopMove = 1;                          //  Stop motion
        till (mStopMove = 0);
    end;
    if (CCW_JOG) do                             //  If the input for negative jog is ON
        mRun = N_Constant_Vel;                  //  Negative jog with the value of Pt533 or Pt585
        till (~CCW_JOG);                        //  Until the input for negative jog is released
        mStopMove = 1;                          //  Stop motion
        till (mStopMove = 0);
    end;
else do
    printl/101("Motor is not enabled!");        //  a warning message for disabled motor
    printl/103("Please release jog button before enabling motor.");   //  Ask to release jog command
    till (S_RDY & ~CW_JOG & ~CCW_JOG);          //  Until the motor is enabled and the jog input is released
end;
goto _loop;
ret;
```

1.7.3 使い方

このサンプルコードはデジタル入力を使用してジョグを実行します。表 1.7.3.1 にデジタル入力 I2 と I3 の機能を示します。

表 1.7.3.1 デジタル入力 I2 と I3 の機能

デジタル入力	機能
I2	正方向 jog
I3	負方向 jog

このサンプルコード内のモーション変数は、表 1.7.3.2 に示すように、Thunder の「Test Run」ウィンドウ内の変数を参照します。

表 1.7.3.2 動作変数と「Test Run」ウィンドウの変数の関係

モーション変数	「Test Run」ウィンドウで変数を参照する
速度	Pt533 (回転), Pt585 (リニア)
加速度	Pt534
減速度	Pt537

注意: Jog を実行する前に変数を設定する必要があります。

1.7.4 アプリケーションの説明

このサンプルコードの冒頭で、#define CW_JOG I2 マクロを使用して、CW_JOG が I2 を指すようにしています。この方法により、プログラムの柔軟性が向上します。このサンプルコードの正のジョグ (CW_JOG) を例にとります。入力を I2 から I8 に変更する場合は、#define CW_JOG I2 を#define CW_JOG I8 に変更するだけです。

このサンプルコードでは、表 1.7.4.1 に示す 2 つの組み込み変数が使用されています。

表 1.7.4.1 2 つの組み込み変数の説明

組み込み変数	説明
mRun	ジョグの方向を設定します。
mStopMove	モーターを停止するには 1 に設定します。

1.8 アナログ入力によるジョグ動作

1.8.1 基本情報

サンプル番号	8
ファイル名	8.pdl
ドライバーモデル	E シリーズドライバー
モータータイプ	AC サーボモーター、リニアモーター、ダイレクトドライブモーター、トルクモーター
ソフトウェアおよびファームウェアの要件	E1 シリーズドライバー: Thunder 1.6.$.$　MDP 2.6.$ 以上 E2 シリーズドライバー: Thunder 1.9.$.$　MDP 3.9.$ 以上
適用モード	内部位置モード
機能	アナログ電圧入力 (0V～+10V) を使用してモーターの速度を制御し、デジタル入力を使用して動作の方向を制御します。
使用する入力	I2, I3, Ref+, Ref-
使用する出力	なし

注: このサンプルプログラムを使用する前に、Thunder のメインウィンドウに移動し、メニューバーで [Tools] を選択して [IO configuration] をクリックし、バッティングを避けるために I2 と I3 を [Not configure] に設定してください。

1.8.2 サンプルプログラム

```
// ============================ Macro ============================
#define LINEAR_RATE_SPEED    1500          //  Set linear motor's rated velocity (Unit: mm/s)
#define AC_RATE_SPEED    3000              //  Set AC servo motor's rated velocity (Unit: rpm)
#define DD_RATE_SPEED    600               //  Set direct drive motor's / torque motor's
                                           //  rated velocity (Unit: rpm)
#float value_temp, rate_speed, speedPervolt;
proc pdl_en() do                           //  Enable the motor
    mEnable = 1;
    till (S_RDY);
end;
proc pdl_dis() do                          //  Disable the motor
    mEnable = 0;
    till (~S_RDY);
end;
// ======================= Unit Conversion proc =======================
```

```
proc Unit_Transform_Vel(float *velTemp) do
    *velTemp /= 100;                          //  0.01V -> 1V
    value_temp = *velTemp;
    if (dMotorType=0) do                      //  the motor is linear motor
        rate_speed = LINEAR_RATE_SPEED;
    end;
    if (dMotorType=1) do                      //  the motor is direct drive motor / torque motor
        rate_speed = DD_RATE_SPEED;
    end;
    if (dMotorType=2) do                      //  the motor is AC servo motor
        rate_speed = AC_RATE_SPEED;
    end;
    speedPervolt = rate_speed / value_temp;
    *velTemp = speedPervolt;
end;
// ======================== Main Program ========================
#task/1;                                      //  Create task #1
#float JOG_VEL_VOL, JogVel_Scale;
mTestRun = 1;                                 //  Forcibly switch to internal position test run mode
mAdjCmd |= 0x1001;                            //  Modify analog offset
sleep 600;
if (dAdjState=1) do
    sleep 300;
    printl/101("Modify analog offset failed!");
    goto _task_end;
else do
    sleep 300;
    printl/103("Modify analog offset succeeded.");
end;
JogVel_Scale = Pt300;                         //  The default of Pt300 is 6V/Rated speed.
Unit_Transform_Vel(&JogVel_Scale);            //  Do unit conversion proc
sleep 1000;
pdl_en();
_loop:
till (S_RDY);                                 //  Wait for servo ready
//   Read voltage from Ref+/- and transform it to reference velocity
JOG_VEL_VOL = dVcmd*JogVel_Scale;
if (dMotorType<>0) do
    Pt533 = JOG_VEL_VOL;
```

```
else do
    Pt585 = JOG_VEL_VOL;
end;
if ( I2 ) do
    if ( I3 ) do                    //  (I2, I3) = (1, 1)
        mRun = 0;                   //  Stop motion
    else do                         //  (I2, I3) = (1, 0)
        mRun = 1;                   //  Positive jog
    end;
else do
    if ( I3 ) do                    //  (I2,I3) = (0,1)
        mRun = -1;                  //  Negative jog
    else do                         //  (I2,I3) = (0,0)
        mRun = 0;                   //  Stop motion
    end;
end;
goto _loop;
_task_end:
ret;
```

1.8.3 使い方

このサンプルコードでは、組み込み変数 dVcmd を使用してアナログ電圧入力を取得し、動作速度を制御します。また、デジタル入力 I2 と I3 を使用して動作方向 (正、負、または停止) を制御します。表 1.8.3.1 に、デジタル入力 I2 と I3 の機能を示します。

表 1.8.3.1 デジタル入力 I2 と I3 の機能

I2	I3	機能
0	0	動作停止
0	1	負方向 jog
1	0	正方向 jog
1	1	動作停止

入力電圧と基準速度のデフォルトの比率は、6V/定格速度です。ユーザーは、Thunder の Pt300 の値を変更することで、この比率を変更できます。

注意：

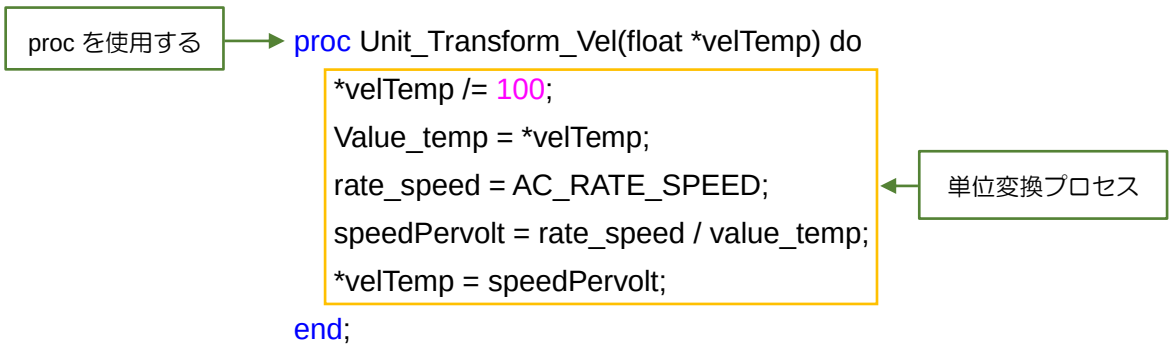
「Rated speed」の値を確認するには、Thunder の「Configuration Wizard」ウィンドウに移動し、「Motor Setup」タブを選択します。

1.8.4 アプリケーションの説明

このサンプルコードでは、アナログ電圧 (Ref+ と Ref- からドライバーに入力され、変数 dVcmd で取得できます) に比率 (入力電圧と基準速度) を掛けると、目的の速度になります。リニアモーターのデフォルトの定格速度は 1500mm/s なので、このサンプルコードで使用すると、0 ～ 6V の電圧で 0 ～ 1500mm/s の速度が得られます。

さらに、このサンプルコードでは、単位変換を行うために PDL の proc スキームが使用されています (セクション 1.4 を参照)。メインプログラムの前に proc を使用し、proc 内に単位変換のプロセスをプログラムするだけです。すると、ユーザーはメインプログラム内で単位変換のためにこの proc を呼び出すことができます。proc は何度も呼び出すことができます。これにより、メインプログラムの複雑さが軽減されます。

Step 1. メインプログラムの前にプロシージャを使用し、プロシージャ内の単位変換のプロセスを編集します。



Step 2. メインプログラムで単位変換が必要になるたびに proc を呼び出します。

(このページはブランクになっています)

2. 変数

この章では、モーション制御変数 (read / write)、モーション状態変数 (read)、物理量監視変数 (read)、サーボ信号状態変数 (read)、サーボ信号 IO 変数 (read) など、PDL を編集およびコンパイルするための E シリーズドライバーの関連変数をユーザーに提供します。

2.1 モーション制御変数（read / write）

名称	mEnable	モード	All	単位	-
タイプ	Int16	範囲	0 ~ 1		
説明	モーターを有効/無効にする				
定義	0 -モーターを無効にする 1 -モーターを有効にする				

サンプル	mEnable = 1; // Enable the motor
	till (S_RDY); // Wait for the motor to be enabled
	mEnable = 0; // Disable the motor
	till (~S_RDY); // Wait for the motor to be disabled				

名称	mTestRun	モード	All	単位	-
タイプ	Unit16	範囲	0 ~ 2		
説明	テスト実行モードを切り替える				
定義	0 -スイッチなし 1 -内部位置テスト実行モードに切り替える 2 -内部速度テスト実行モードに切り替える				

サンプル	mTestRun = 1; // Forcibly switch to internal position test run mode
	till (dOperMode = 0); // Ensure it is internal position mode
	mMovePos = 100;
	sleep 1; // Wait for the delay time of status transition
	till (COIN); // Wait until the motor is in-position
	mTestRun = 2; // Forcibly switch to internal velocity test run mode
	till (dOperMode = 4); // Ensure it is internal velocity mode
	mJog = 1; // Positive constant velocity motion
	sleep 1; // Wait for the delay time of status transition
	till (V_CMP); // Wait until the pre-defined velocity is reached				

名称	mMovePos	モード	内部ポジション	単位	control unit
タイプ	Int32	範囲	-2147483648 ~ 2147483647		
説明	モーターを指定された位置に移動します				
定義	値 - 指定された位置				

サンプル	Pt533 = 100; // Set rotary motor motion velocity as 100 rpm
	mTestRun = 1; // Forcibly switch to internal position test run mode
	mMovePos = 100; // Move the motor to the position of 100 control unit
	// Rotary motor motion velocity is determined by the setting of Pt533.
	// Linear motor motion velocity is determined by the setting of Pt585.
	// Motor motion acceleration time is determined by the setting of Pt534.
	// Motor motion deceleration time is determined by the setting of Pt537.
	sleep 1; // Wait for the delay time of status transition
	till (COIN); // Wait until the motor is in-position
	// Attention: Do not simultaneously give “mMovePos” and “mRun” commands				

名称	mRun	モード	内部ポジション	単位	-
Type	Int16	範囲	-1 ~ 1		
説明	内部位置テスト実行モードでの一定速度動作				
定義	0 -動作停止 1 -正の等速モーション -1 -負の等速モーション				

サンプル	mTestRun = 1; // Forcibly switch to internal position test run mode
	mRun = 1; // Positive constant velocity motion
	mRun = -1; // Negative constant velocity motion
	mRun = 0; // Motion stop
	// Rotary motor motion velocity is determined by the setting of Pt533.
	// Linear motor motion velocity is determined by the setting of Pt585.
	// Motor motion acceleration time is determined by the setting of Pt534.
	// Motor motion deceleration time is determined by the setting of Pt537.
	// Attention: Do not simultaneously give “mMovePos” and “mRun” commands				

名称	mStopMove	モード	内部ポジション	単位	-
タイプ	Uint16	範囲	0 ~ 2		
説明	モーターを停止する				
定義	0 –影響がない、またはモーターが停止している 1 -緊急減速速度でモーターを停止する（Pt538） 2 -減速速度でモーターを停止する（Pt537）				

サンプル	mTestRun = 1; // Forcibly switch to internal position test run mode
	mMovePos = 10000; // Move the motor to the position of 10000 control unit
	till (I1); // Ensure I1 is triggered
	// (assume that I1 is the stop motion input set by users)
	mStopMove = 1;
	// The motor stops moving.
	// Motor motion deceleration time is determined by the setting of Pt538.				

名称	mJog	モード	内部速度	単位	-
タイプ	Int16	範囲	-1 ~ 1		
説明	内部速度テスト実行モードでの一定速度動作				
定義	0 -動作停止 1 -正の等速モーション -1 -負の等速モーション				

サンプル	mTestRun = 2; // Forcibly switch to internal velocity test run mode
	mJog = 1; // Positive constant velocity motion
	mJog = -1; // Negative constant velocity motion
	mJog = 0; // Motion stop
	// Rotary motor motion velocity is determined by the setting of Pt304.
	// Linear motor motion velocity is determined by the setting of Pt383.
	// Motor motion acceleration time is determined by the setting of Pt305.
	// Motor motion deceleration time is determined by the setting of Pt306.				

名称	mHome	モード	All	単位	-
タイプ	Int16	範囲	0 ~ 2		
説明	ホーミングを実行する				
定義	0 -ホーミングを実行しない 1 -ホーミングを実行する 2 -ホーミング実行を停止する				

サンプル	mHome = 0; // Do not execute homing
	mHome = 1; // Execute homing
	till (dHomeState = 2);				

名称	mAdjCmd	モード	All	単位	-
タイプ	Uint16	範囲	0 ~ 65535		
説明	アクションコマンドの調整				
定義	0 -影響なし 0x1001 –アナログオフセットを変更する 0x1008 -マルチターン数をクリアする				

サンプル	mAdjCmd = 0x1008;
	sleep 600; // The execution time is about 600 ms
	if (dAdjState = 1) do
	printl/101("Adjusted action failed!");
	else do
	printl/103("Adjusted action succeeded.");
	end;				

名称	mInInvert	モード	All	単位	-
タイプ	Uint16	範囲	0 ~ 65535		
説明	デジタル入力信号反転				
定義	各ビットは対応する入力論理反転を表す。 0 -影響なし 0x0001 - I1 反転 0x0002 - I2 反転 0x0004 - I3 反転 0x0005 - I1、I3 反転 など。				

サンプル	mInInvert = 1; // I1 invert
	till (I1); // Ensure I1 is triggered				

名称	mOutInvert	モード	All	単位	-
タイプ	Uint16	範囲	0 ~ 65535		
説明	デジタル出力信号反転				
定義	各ビットは対応する出力論理反転を表す。 0 -影響なし 0x0001 - O1 反転 0x0002 - O2 反転 0x0004 - O3 反転 0x0005 - O1、O3 反転 など。				
サンプル	mOutInvert = 1; // O1 invert				

2.2 モーションステータス変数（read）

名称	dMotorType	モード	All	単位	-
タイプ	Int16	範囲	0 ~ 2		
説明	モータータイプ表示				
定義	0 – リニアモーター 1 -ダイレクトドライブモーター / トルクモーター 2 - AC サーボモーター				

名称	dCntPerUnit	モード	All	単位	下記参照
タイプ	Int32	範囲	0 ~ 2147483648		
説明	エンコーダー分解能単位表示				
単位の説明	リニアモーター：count /100mm ダイレクトドライブモーター / トルクモーター： count / rev AC サーボモーター： count / rev				

名称	dVcmd	モード	All	単位	Volt
タイプ	Float32	範囲	-10 ~ 10		
説明	速度指令電圧表示				

名称	dTcmd	モード	All	単位	Volt
タイプ	Float32	範囲	-10 ~ 10		
説明	トルク指令電圧表示				

名称	dHomeState	モード	All	単位	-
タイプ	Int16	範囲	-1 ~ 4		
説明	ホーミング状態表示				
定義	0 –ホーミングが実行されない 2 -ホーミング成功 3 -ホーミング 4 -ホーミング停止 -1 -ホーミング失敗				

名称	dOperMode	モード	All	単位	-
タイプ	Int16	範囲	0 ~ 4		
説明	操作モード表示				
定義	0 -内部位置モード 1 -位置モード 2 -速度モード 3 -トルクモード 4 -内部速度モード				

名称	X_run	モード	位置 内部位置	単位	-
タイプ	State	範囲	-		
説明	モーター動作出力信号				

名称	dAdjState	モード	All	単位	-
タイプ	Uint16	範囲	0 ~ 1		
説明	アクション表示の調整				
定義	mAdjCmd の設定結果。 0 -成功 1 -失敗 225 -実行中				

2.3 物理量モニタリング変数（read）

物理量の名称説明については、「E1 シリーズドライバーユーザーマニュアル」の図 11.3.1.1 および「E2 シリーズドライバーユーザーマニュアル」の図 11.3.1.1 を参照してください。それらの動作は、「E1 シリーズドライバーユーザーマニュアル」および「E2 シリーズドライバーユーザーズマニュアル」の「範囲 - 物理量」のモニタリング項目と同じです。

物理量 (1)	dPosErr	タイプ	Int32	可変単位	control unit
説明	位置エラー				

物理量 (2)	dFbPos	タイプ	Int32	可変単位	control unit
説明	フィードバック位置				

物理量 (3)	dPosRefVel	タイプ	Float32	可変単位	rpm または mm/s
説明	位置基準速度 注: この変数は、E1 / E2 ファームウェア バージョン 2.10.6 / 3.10.6 以下には使用できません。				

物理量 (4)	dDLPosErr	タイプ	Int32	可変単位	control unit
説明	モーター負荷位置偏差				

物理量 (5)	dVelFFCmd	タイプ	Float32	可変単位	rpm または mm/s
説明	速度フィードフォワード				

物理量 (6)	dVelCmd	タイプ	Float32	可変単位	rpm または mm/s
説明	基準速度				

物理量 (7)	dFbVel	タイプ	Float32	可変単位	rpm または mm/s
説明	モーター速度				

物理量 (8)	dToqFFCmd	タイプ	Float32	可変単位	A-amp
説明	トルクフィードフォワード 注：E1 / E2 ファームウェアバージョンが 2.10.6 / 3.10.6 以下の場合、可変単位は 0.1% ドライバーピーク電流です。				

物理量 (9)	dToqCmd	タイプ	Float32	可変単位	A-amp
説明	トルク参考 注: E1 / E2 ファームウェアバージョンが 2.10.6 / 3.10.6 以下の場合、可変単位は 0.1% ドライバーピーク電流です。				

物理量 (10)	dCurrCmd	タイプ	Float32	可変単位	A-amp
説明	コマンド電流 注：E1 / E2 ファームウェアバージョンが 2.10.6 / 3.10.6 以下の場合、可変単位は 0.1% ドライバーピーク電流です。				

物理量 (11)	dFbCurr	タイプ	Float32	可変単位	A-amp
説明	モーター電流 注: E1 / E2 ファームウェアバージョンが 2.10.6 / 3.10.6 以下の場合、この変数はモーター電流の平方根を示し、単位は A-amp^2 です。				

物理量 (12)	dServoVoltPercent	タイプ	Float32	可変単位	%
説明	サーボ電圧パーセンテージ 注: この変数は、E1 / E2 ファームウェアバージョン 2.10.6 / 3.10.6 以下には使用できません。				

物理量 (13)	dHallSignal	タイプ	Int16	可変単位	-
説明	デジタルホールセンサー				

物理量 (14)	dMotLoadPercent	タイプ	Float32	可変単位	%
説明	モーター過負荷率 注: この変数は、E1 / E2 ファームウェア バージョン 2.10.6 / 3.10.6 以下には使用できません。				

物理量 (15)	dAmpPosErr	タイプ	Int32	可変単位	count
説明	ポジションアンプエラー				

物理量 (16)	dVelErr	タイプ	Float32	可変単位	rad/s または mm/s
説明	速度エラー				

物理量 (17)	dMasterPos	タイプ	Int32	可変単位	control unit
説明	マスターフィードバック位置 注: この変数はマスター軸ドライバーでのみ使用できます。				

物理量 (18)	dSlavePos	タイプ	Int32	可変単位	control unit
説明	スレーブフィードバック位置 注: この変数はスレーブ軸ドライバーでのみ使用できます。				

物理量 (19)	dYawPos	タイプ	Int32	可変単位	control unit
説明	ヨー位置				

物理量 (20)	X_run_pcmd	タイプ	State	可変単位	-
説明	実行位置コマンドの状態				

物理量 (21)	dEffectGain	タイプ	Float32	可変単位	-
説明	有効ゲイン				
Definition	0 -ゲイン切り替えなし 1 -最初の有効ゲインを使用する 2 - 2 番目の有効ゲインを使用する				

物理量 (22)	dFbPosInt	タイプ	Int32	可変単位	control unit
説明	内部フィードバック位置				

物理量 (23)	dLinearCmd	タイプ	Int32	可変単位	A-amp
説明	ガントリーリニアコマンド電流				

物理量 (24)	dYawCmd	タイプ	Int32	可変単位	A-amp
説明	ガントリーヨーコマンド電流				

物理量 (25)	dYawPosErr	タイプ	Int32	可変単位	control unit
説明	ガントリーヨー位置エラー				

物理量 (26)	dLoadPos_MultiMotion	タイプ	Int32	可変単位	control unit
説明	負荷側シングルターン位置 (マルチモーションのみ) 注: この変数は、E1 / E2 ファームウェア バージョン 2.10.6 / 3.10.6 以下には使用できません。				

物理量 (27)	dLoadPos	タイプ	Int32	可変単位	control unit
説明	負荷側の位置 注: この変数は、E1 / E2 ファームウェア バージョン 2.10.6 / 3.10.6 以下には使用できません。				

物理量 (28)	dMotPeakLoadPercent	タイプ	Float32	可変単位	%
説明	ピーク負荷率 注: この変数は、E1 / E2 ファームウェア バージョン 2.10.6 / 3.10.6 以下には使用できません。				

物理量 (29)	dRegenLoadPercent	タイプ	Float32	可変単位	%
説明	回生負荷率 注: この変数は、E1 / E2 ファームウェア バージョン 2.10.6 / 3.10.6 以下には使用できません。				

物理量 (30)	dDrvLoadPercent	タイプ	Float32	可変単位	%
説明	ドライバーの過負荷率 注: この変数は、E1 / E2 ファームウェア バージョン 2.10.6 / 3.10.6 以下には使用できません。				

物理量 (31)	dBusVolt	タイプ	Float32	可変単位	volt
説明	バス電圧 注: この変数は、E1 / E2 ファームウェア バージョン 2.10.6 / 3.10.6 以下には使用できません。				

物理量 (32)	dExtPosCmd	タイプ	Int32	可変単位	control unit
説明	外部位置コマンド 注: この変数は、E1 / E2 ファームウェア バージョン 2.10.6 / 3.10.6 以下には使用できません。				

物理量 (33)	dInrPosCmd	タイプ	Int32	可変単位	control unit
説明	内部位置コマンド 注: この変数は、E1 / E2 ファームウェア バージョン 2.10.6 / 3.10.6 以下には使用できません。				

2.4 サーボ信号ステータス変数（read）

サーボ信号ステータス変数のパフォーマンスは、「E1 シリーズドライバーユーザーマニュアル」および「E2 シリーズドライバーユーザーマニュアル」の「スコープ - サーボ信号ステータス」のモニタリング項目と同じです。

モニタリング項目 No.	変数名	説明
51	S_ON	サーボオン入力信号
52	P_CON	比例制御入力信号
53	P_OT	前方禁止入力信号
54	N_OT	逆方向禁止入力信号
55	ALM_RST	アラームリセット入力信号
56	P_CL	前方外部トルク制限入力信号
57	N_CL	逆方向外部トルク制限入力信号
58	C_SEL	制御方式スイッチング入力信号
59	SPD_D	モーター回転方向入力信号
60	SPD_A	内部設定速度入力信号
61	SPD_B	内部設定速度入力信号
62	ZCLAMP	ゼロクランプ入力信号
63	INHIBIT	コマンドパルス抑制入力信号
64	G_SEL	ゲイン切替入力信号
65	PSEL	コマンドパルス乗算スイッチング入力信号
66	RST	ドライバーリセット入力信号
67	DOG	原点近傍センサー入力信号
68	HOM	ドライバー内蔵ホーミング手順入力信号
69	MAP	ドライバーエラーマップ入力信号
70	FSTP	強制停止入力信号
71	CLR	位置偏差クリア入力信号
72	ALM	警報出力信号
73	COIN	位置決め完了出力信号
74	V_CMP	速度到達出力信号
75	TGON	回転検知/移動検知出力信号
76	D_RDY	ドライバー準備完了出力信号
77	S_RDY	サーボレディ出力信号
78	CLT	トルク制限検出出力信号
79	VLT	速度制限検出出力信号
80	BK	ブレーキ制御出力信号

モニタリング項目 No.	変数名	説明
81	WARN	警告出力信号
82	NEAR	位置決め近傍信号出力
83	PSELA	コマンドパルス乗算スイッチング出力信号
84	PT	位置トリガーデジタル出力信号
85	DBK	外部ダイナミックブレーキ出力信号
86	HOMED	ドライバーホーミング完了出力信号
87	PAO	エンコーダー分割パルス出力信号-A 相
88	PBO	エンコーダー分割パルス出力信号-B 相
89	PZO	エンコーダー分割パルス出力信号-Z 相
90	X_ind	インデックス信号

2.5 サーボ信号 IO 変数（read）

Variable name	Description
I1	デジタル入力信号 1
I2	デジタル入力信号 2
I3	デジタル入力信号 3
I4	デジタル入力信号 4
I5	デジタル入力信号 5
I6	デジタル入力信号 6
I7	デジタル入力信号 7
I8	デジタル入力信号 8
I9	デジタル入力信号 9
I10	デジタル入力信号 10
O1	デジタル出力信号 1
O2	デジタル出力信号 2
O3	デジタル出力信号 3
O4	デジタル出力信号 4
O5	デジタル出力信号 5

サンプル

```
till (I1);                    //   Wait until I1 is triggered (same as other input signal)
seton O1;
//   If there is no output condition for O1, users can change output status via PDL.
//   (same as other output signal)
```

E シリーズドライバーの PDL 集
バージョン：V2.2　　2024 年 7 月改訂

１．HIWIN は HIWIN Mikrosystem Corp., HIWIN Technologies Corp., ハイウィン株式会社の登録商標です。ご自身の権利を保護するため、模倣品を購入することは避けてください。

２．実際の製品は、製品改良等に対応するため、このカタログの仕様や写真と異なる場合があります。

３．HIWIN は「貿易法」および関連規制の下で制限された技術や製品を販売・輸出しません。制限された HIWIN 製品を輸出する際には、関連する法律に従って、所管当局によって承認を受けます。また、核・生物・化学兵器やミサイルの製造または開発に使用することは禁じます。
