

PCL6125-EB_ApplicationSample

－解説編－

Apr.25.2025

営業本部

NPM

顧客「満足」から「感動」へ。

日本パルスモーター株式会社

<http://www.pulsemotor.com>

1. ApplicationSample ファイル構成
2. ソリューションファイル PCL6125-EB_Sample.sln
3. ソースコード Form1.cs
4. PCL6125アクセス関数 accessPCL.sc

1. ApplicationSample ファイル構成

◆上位CPUとのバス接続I/F

PCL61x5シリーズは、上位CPUとシリアルバス(SPI)とパラレルバスの2種類の接続方法が用意されています。
Pcl6125-EB評価ボードでは、シリアルバスで接続されていますのでこのサンプルプログラムもそれに対応した仕様となっています。

◆PCL6125-EB_ApplicationSample_V***JE のファイル構成

このプログラムは、統合開発環境「Microsoft Visual Studio Express 2013 for Windows Desktop (無償版)」によって作成されており、プログラミング言語「Microsoft Visual C#」を使って記述されています。
このテキストではモーション制御に関するソースコードForm1.cs、accessPCL.csの説明を行います。

¥PCL6125-EB_ApplicationSample_V***JE ***: Version番号

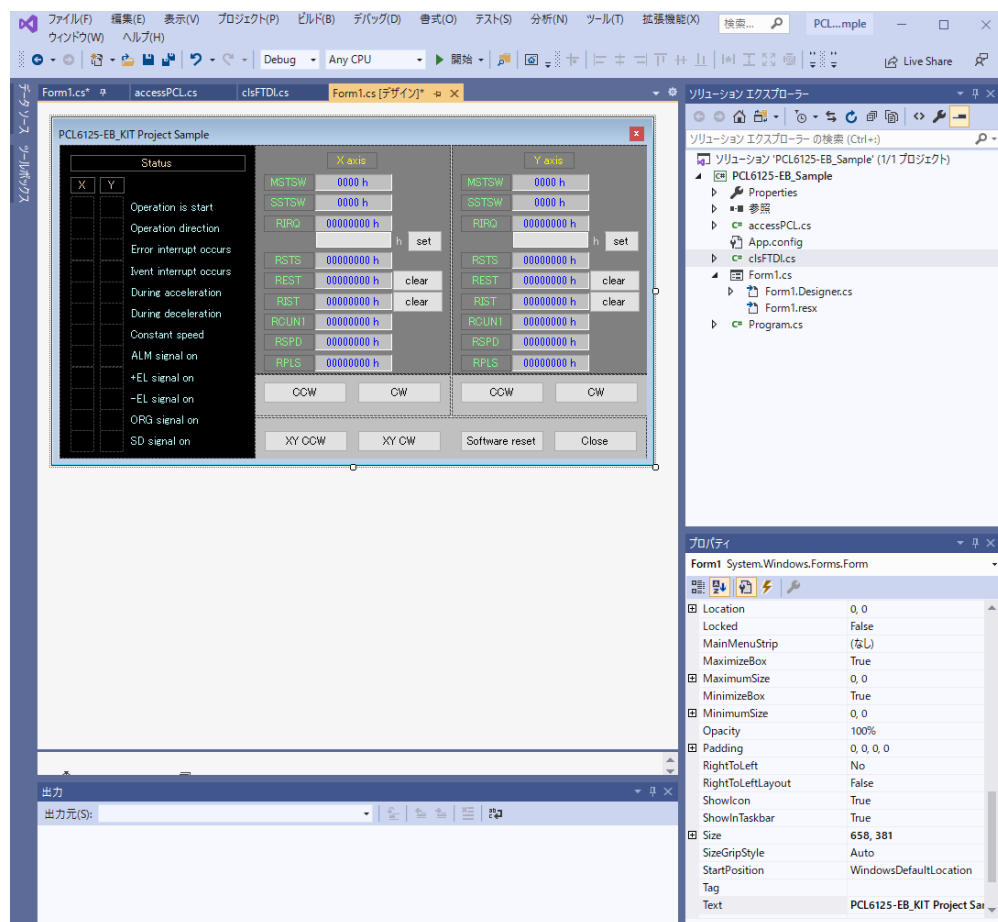
```
|
|----¥Driver
|      |----CDM21226_Setup.exe      .... USBデバイスドライバインストーラ(FTDI製)
|
|----¥PCL6125-EB_Sample
|      |----¥bin
|          |----¥Debug
|              |----PCL6125-EB_Sample.exe      .... 実行ファイル
|              |----FTD2XX_NET.dll             .... FTDI製FT232HL_ライブラリ (実行時必須)
|              |----FTD2XX_NET.xml             .... FTDI製FT232HL_xmlファイル (実行時不要)
|              |----その他
|
|          |----Form1.cs                    .... ソースコード
|          |----accessPCL.cs                .... NPM製PCL6125_アクセス関数ソースコード
|          |----clsFTDI.cs                  .... FTDI製FT232HL_アクセス関数ソースコード
|          |----FTD2XX_NET.dll               .... FTDI製FT232HL_ライブラリ
|          |----FTD2XX_NET.XML               .... FTDI製FT232HL_xmlファイル
|          |----その他
|
|----PCL6125-EB_Sample.sln      .... ソリューションファイル
|----その他
```

2. ソリューションファイル PCL6125-EB_Sample.sln

ソリューションファイルのコード表示、実行するには統合開発環境「Microsoft Visual Studio」が必要ですが、単に内容を表示するだけでしたらテキストエディターの「メモ帳」でも見ることができます。

また無償版「Microsoft Visual Studio Express」の他に無償版「Microsoft Visual Studio community」も有りますが、どちらも無償使用に条件がありますのであらかじめ確認が必要です。

◆ソリューションファイル:PCL6125-EB_Sample.sln



3. ソースコード Form1.cs

◆ソースコード:Form1.cs 操作画面

ステータス

表記	内容		
Operation is start	動作中	MSTSW	SSCM
Operation direction	動作方向 (CW:0,CCW:1)	RSTS	SDIR
Error interrupt occurs	エラー割り込み発生	MSTSW	SERR
Ivent interrupt occurs	イベント割り込み発生	MSTSW	SINT
During acceleration	加速中	SSTSW	SFU
During deceleration	減速中	SSTSW	SFD
Constant speed	定速中	SSTSW	SFC
ALM signal on	ALM信号オン	SSTSW	SALM
+EL signal on	+EL信号オン	SSTSW	SPEL
-EL signal on	-EL信号オン	SSTSW	SMEL
ORG signal on	ORG信号オン	SSTSW	SORG
SD signal on	SD信号オン	SSTSW	SSD

レジスタ

表記	内容
MSTSW	メインステータス
SSTSW	サブステータス
RIRQ	イベント割込み要因設定レジスタ
RSTS	拡張ステータス
REST	エラー割込み要因レジスタ
RIST	イベント割込み要因レジスタ
RCUN1	カウンタ1
RSPD	現在速度モニター
RPLS	位置決めカウンタ取得

起動

ボタン	内容
CCW	CCW方向起動、2304pulse
CW	CW方向起動、2304pulse
XYCCW	X、Y軸同時にCCW方向起動、2048pulse
XYCW	X、Y軸同時にCW方向起動、2048pulse
Software reset	PCL6125をリセット
set	RIRQへのデータ設定
clear	REST、RISTをクリア
Close	ソフト終了

3. ソースコード Form1.cs

◆ソースコード:Form1.cs 内容説明

Form1.csの内容を確認するに当たって、必要なPCL6125アクセス関数:accessPCL.scの構文

・ Write_REG レジスタへの書き込み関数

```
////////////////////////////////////  
/// <summary>  
/// レジスタの書き込み  
/// Write register  
/// RegD : 書き込みデータ  
/// Write data  
/// Jsc : 軸選択コマンド  
/// Axis selection command  
/// comm : レジスタライトコマンド  
/// Register write command  
/// </summary>  
/// <param name="description"></param>  
/// <returns></returns>  
13 個の参照  
public void Write_REG(ref byte[] FtBuff, ref int FtIndex, uint RegD, byte Jsc, byte comm)
```

・ Read_REG レジスタの読出し関数

```
////////////////////////////////////  
/// <summary>  
/// レジスタの読出し  
/// Read registers  
/// comm : レジスタリードコマンド  
/// Register read command  
/// </summary>  
/// <param name="description"></param>  
/// <returns></returns>  
7 個の参照  
public void Read_REG(ref byte[] FtBuff, ref int FtIndex, byte comm)
```

・ Write_COM 動作コマンドの書き込み関数

```
////////////////////////////////////  
/// <summary>  
/// コマンドの書き込み  
/// Write command  
/// Jsc : 軸選択コマンド  
/// Axis selection command  
/// comm : コマンド  
/// Command  
/// </summary>  
/// <param name="description"></param>  
/// <returns></returns>  
3 個の参照  
public void Write_COM(ref byte[] FtBuff, ref int FtIndex, byte Jsc, byte comm)
```


3. ソースコード Form1.cs

◆ソースコード: Form1.cs 初期設定

```

517  // <summary>
518  /// 初期設定
519  /// Initial setting
520  /// </summary>
521  private void InitSet()
522  {
523      FtIndex = 0;
524
525      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000002, 0x01, 0x9C); // write RENV1 (X-Axis)
526      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x80000055, 0x01, 0x9D); // write RENV2 (X-Axis)
527      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x000004AF, 0x01, 0x85); // write PRMG (X-Axis)
528      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000041, 0x01, 0x87); // write PRMD (X-Axis)
529
530      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000002, 0x02, 0x9C); // write RENV1 (Y-Axis)
531      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x80000055, 0x02, 0x9D); // write RENV2 (Y-Axis)
532      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x000004AF, 0x02, 0x85); // write PRMG (Y-Axis)
533      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000041, 0x02, 0x87); // write PRMD (Y-Axis)
534      //
535      ftStatus = hAPCL.SendUsb(ref FtBuff, ref FtIndex);
536  }
537

```

accessPCL.sc、レジスタへの書き込み関数
 0x00000002: データ
 0x01: 1軸目(X軸)
 0x9C: コマンド(環境設定1: RENV1への書き込み)

0x02: 2軸目(Y軸)

項目		記号	内容	設定値、指令値		レジスタ書き込みコマンド	
				dec	hex	記号	COMB
環境設定	環境設定1	RENV1			00000002h	WRENV1	9Ch
	環境設定2	RENV2			80000055h	WRENV2	9Dh
移動量・速度	速度倍率設定	PRMG	1倍	1199	04AFh	WPRMG	85h
動作モード	動作モード	RMD	相対移動		00000041h	WPRMD	87h

3. ソースコード Form1.cs

◆ソースコード: Form1.cs X軸CW・CCW動作

```

316  /// <summary>
317  /// X軸 CCW処理実行
318  /// X axis CCW process execution
319  /// </summary>
320  private void Exec_XCCW()
321  {
322      FtIndex = 0;
323
324      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000001, 0x01, 0x81); // write PRFL (X-Axis)
325      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000400, 0x01, 0x82); // write PRFH (X-Axis)
326      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00002588, 0x01, 0x83); // write PRUR (X-Axis)
327      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0xFFFF700, 0x01, 0x80); // write PRMV (X-Axis)
328      //
329      hAPCL.Write_COM(ref FtBuff, ref FtIndex, 0x01, 0x53); // write High speed start 2 (X-Axis)
330      //
331      ftStatus = hAPCL.SendUsb(ref FtBuff, ref FtIndex);
332  }
333
334  /// <summary>
335  /// X軸 CW処理実行
336  /// X axis CW process execution
337  /// </summary>
338  private void Exec_XCW()
339  {
340      FtIndex = 0;
341
342      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000001, 0x01, 0x81); // write PRFL (X-Axis)
343      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000400, 0x01, 0x82); // write PRFH (X-Axis)
344      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00002588, 0x01, 0x83); // write PRUR (X-Axis)
345      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000900, 0x01, 0x80); // write PRMV (X-Axis)
346      //
347      hAPCL.Write_COM(ref FtBuff, ref FtIndex, 0x01, 0x53); // write High speed start 2 (X-Axis)
348      //
349      ftStatus = hAPCL.SendUsb(ref FtBuff, ref FtIndex);
350  }

```

項目		記号	内容	設定値、指令値		レジスタ書き込みコマンド	
				dec	hex	記号	COMB
移動量・速度	FL速度設定	PRFL	1pps	1	00000001h	WPRFL	81h
	FH速度設定	PRFH	1024pps	1024	00000400h	WPRFH	82h
	加速レート設定	PRUR	1000ms	9608	00002588h	WPRUR	83h
CCW動作	移動量(目標位置)	PRMV	-2304puls	-2304	FFFFFF700h	WPRMV	80h
CW動作	移動量(目標位置)	PRMV	2304puls	2304	00000900h	WPRMV	80h

起動	動作コマンド	STAUD	高速スタート2(加減速)	53h
----	--------	-------	--------------	-----

0x01:1軸目(X軸)

* Y軸(2軸目)の場合は、0x02

3. ソースコード Form1.cs

◆ソースコード: Form1.cs XY軸CW・CCW動作

```

388  /// <summary>
389  /// X,Y軸 CCW処理実行
390  /// X,Y axis CCW process execution
391  /// </summary>
392  private void Exec_XYCCW()
393  {
394      FtIndex = 0;
395
396      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000001, 0x01, 0x81); // write PRFL (X-Axis)
397      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000400, 0x01, 0x82); // write PRFH (X-Axis)
398      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00002588, 0x01, 0x83); // write PRUR (X-Axis)
399      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0xFFFF700, 0x01, 0x80); // write PRMV (X-Axis)
400
401      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000001, 0x02, 0x81); // write PRFL (Y-Axis)
402      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000400, 0x02, 0x82); // write PRFH (Y-Axis)
403      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00002588, 0x02, 0x83); // write PRUR (Y-Axis)
404      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0xFFFF700, 0x02, 0x80); // write PRMV (Y-Axis)
405      //
406      hAPCL.Write_COM(ref FtBuff, ref FtIndex, 0x03, 0x53); // write High speed start 2 (X,Y-Axis)
407      //
408      FtStatus = hAPCL.SendUsb(ref FtBuff, ref FtIndex);
409  }
410
411  /// <summary>
412  /// X,Y軸 CW処理実行
413  /// X,Y axis CW process execution
414  /// </summary>
415  private void Exec_XYCW()
416  {
417      FtIndex = 0;
418
419      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000001, 0x01, 0x81); // write PRFL (X-Axis)
420      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000400, 0x01, 0x82); // write PRFH (X-Axis)
421      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00002588, 0x01, 0x83); // write PRUR (X-Axis)
422      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000900, 0x01, 0x80); // write PRMV (X-Axis)
423
424      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000001, 0x02, 0x81); // write PRFL (Y-Axis)
425      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000400, 0x02, 0x82); // write PRFH (Y-Axis)
426      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00002588, 0x02, 0x83); // write PRUR (Y-Axis)
427      hAPCL.Write_REG(ref FtBuff, ref FtIndex, 0x00000900, 0x02, 0x80); // write PRMV (Y-Axis)
428      //
429      hAPCL.Write_COM(ref FtBuff, ref FtIndex, 0x03, 0x53); // write High speed start 2 (X,Y-Axis)
430      //
431      FtStatus = hAPCL.SendUsb(ref FtBuff, ref FtIndex);
432  }

```

項目	記号	内容	設定値、指令値		レジスタ書き込みコマンド	
			dec	hex	記号	COMB
移動量・速度	FL速度設定	PRFL	1pps	00000001h	WPRFL	81h
	FH速度設定	PRFH	1024pps	00000400h	WPRFH	82h
	加速レート設定	PRUR	1000ms	00002588h	WPRUR	83h
CCW動作	移動量(目標位置)	PRMV	-2304puls	FFFFFF700h	WPRMV	80h
CW動作	移動量(目標位置)	PRMV	2304puls	00000900h	WPRMV	80h
起動	動作コマンド	STAUD	高速スタート2(加減速)	53h		

0x01:1軸目(X軸)
 0x02:2軸目(Y軸)
 0x03:1, 2軸目(X,Y軸)同時選択

3. ソースコード Form1.cs

◆ソースコード: Form1.cs Software reset

```
434  /// <summary>  
435  /// ソフトウェアリセット実行  
436  /// Software reset execution  
437  /// </summary>  
438  private void Exec_SoftReset()  
439  {  
440      FtIndex = 0;  
441  
442      hAPCL.Write_COM(ref FtBuff, ref FtIndex, 0x01, 0x04); // write Software reset (X-Axis)  
443      //  
444      ftStatus = hAPCL.SendUsb(ref FtBuff, ref FtIndex);  
445      //  
446      InitSet();  
447  }
```

accessPCL.sc 動作コマンドの書き込み関数
0x01:1軸目
0x04:コマンド(LSIをソフトウェアリセット)

ソフトウェアリセット	04h	SRST	LSIをソフトウェアリセット
------------	-----	------	----------------

3. ソースコード Form1.cs

◆ソースコード: Form1.cs set動作 および clear動作

```
449 /// <summary>  
450 /// レジスタへのライト処理実行  
451 /// Execute write processing to register  
452 /// </summary>  
453 /// <param name="type"></param>  
454 private void Exec_WriteReg(int type, byte axis)  
455 {  
456     uint Data = 0;  
457     byte comm = 0;  
458   
459     switch (type)  
460     {  
461         case 0:  
462             comm = 0xAC;  
463             if (axis == 0x01)  
464             {  
465                 if (this.t_rirq.Text == "")  
466                 {  
467                     Data = 0;  
468                 }  
469             } else  
470             {  
471                 Data = (uint)(Convert.ToInt32(this.t_rirq.Text, 16));  
472             }  
473         }  
474         else  
475         {  
476             if (this.t_rirq_y.Text == "")  
477             {  
478                 Data = 0;  
479             }  
480             else  
481             {  
482                 Data = (uint)(Convert.ToInt32(this.t_rirq_y.Text, 16));  
483             }  
484         }  
485         break;
```

0xAC: イベント割り込み要因設定 (RIRQ)

0xB2: エラー割り込み要因取得 (REST)

0xB3: イベント割り込み要因取得 (RIST)

Axis==0x01 1軸目(X軸)

RENV2レジスタ

Bit31 各種の割り込み要因ビット(MSTS.SENI, REST, RIST)と目標位置以外で
停止状態ビット(MSTS.SEOR)のクリア方法選択: 1 書込みで手動クリア
が設定されているため

エラー割り込み要因取得レジスタ(REST)、RIST(イベント割り込み要因取得レジスタ(RIST))
のクリアは、読んだ値をそのまま書込むことでクリアする。

```
486   
487   
488         case 1:  
489             comm = 0xB2;  
490             if (axis == 0x01)  
491             {  
492                 Data = rest;  
493             }  
494             else  
495             {  
496                 Data = rest_y;  
497             }  
498             break;  
499   
500         case 2:  
501             comm = 0xB3;  
502             if (axis == 0x01)  
503             {  
504                 Data = rist;  
505             }  
506             else  
507             {  
508                 Data = rist_y;  
509             }  
510             break;  
511         }  
512         //  
513         hAPCL.Write_REG(ref FtBuff, ref FtIndex, Data, axis, comm); // write  
514         //  
515         ftStatus = hAPCL.SendUsb(ref FtBuff, ref FtIndex);
```


4. PCL6125アクセス関数 accessPCL.sc

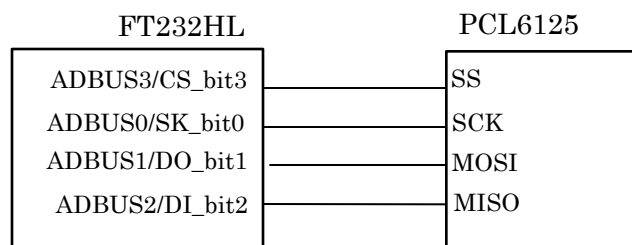
◆シリアルバスアクセス方法（PCL6125-EB評価ボード、FT232HLとPCL6125の接続）

・FT232HL、シリアルバス_SPI CS信号のHigh/Low制御

USB-Multipurpose I/F IC (FTDI製) FT232HLとPCL6125のハードウェア、SPI接続

FT232HLのMPSSE (Multi-Protocol Synchronous Serial Engine) 機能を使用

(取扱説明書「AN2232C-01 Command Processor for MPSSE and MCU Host Bus Emulation Modes」参照)



FT232HL、汎用入出力端子の設定

3.6.1 Set Data bits LowByte

0x08 : LowByte (ADBUSB7-0) のbitをセットするコマンド

0xValue : LowByte (ADBUSB7-0) bit3: CS Low:0

0xDirectionyte : 指定したbitを出力にする 1:出力

・CS: LowにしてPCL6125にデータを書き込む

0x08

0x00 : LowByte (ADBUSB7-0) bit3: CS 0: Low

0x0B : bit3: CS, bit1: DO, bit0: SK ⇒ 1: 出力設定

```
//CS=Low
FtBuff[FtIndex++] = 0x80;
FtBuff[FtIndex++] = 0x00;
FtBuff[FtIndex++] = 0x0B;
```

・CS: HighにしてPCL6125にデータを書き込み終了

0x08

0x08 : LowByte (ADBUSB7-0) bit3: CS 1: High

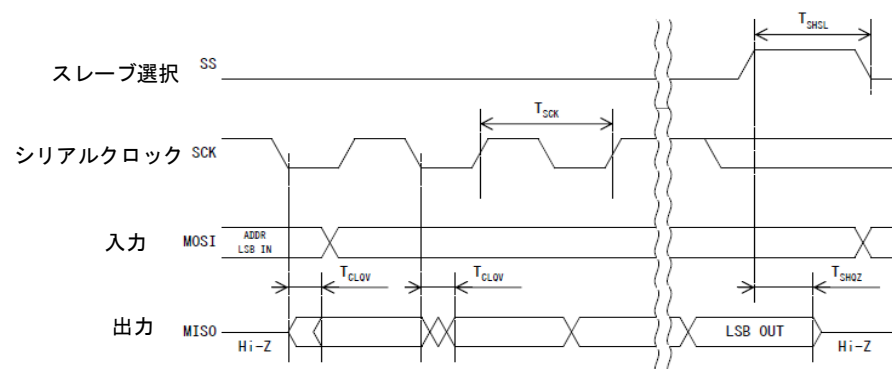
0x0B : bit3: CS, bit1: DO, bit0: SK ⇒ 1: 出力設定

```
//CS=High
FtBuff[FtIndex++] = 0x80;
FtBuff[FtIndex++] = 0x08;
FtBuff[FtIndex++] = 0x0B;
```

【PCL6125_SPI_シリアルバスタイミング】 (PCL61x5取扱説明書より)

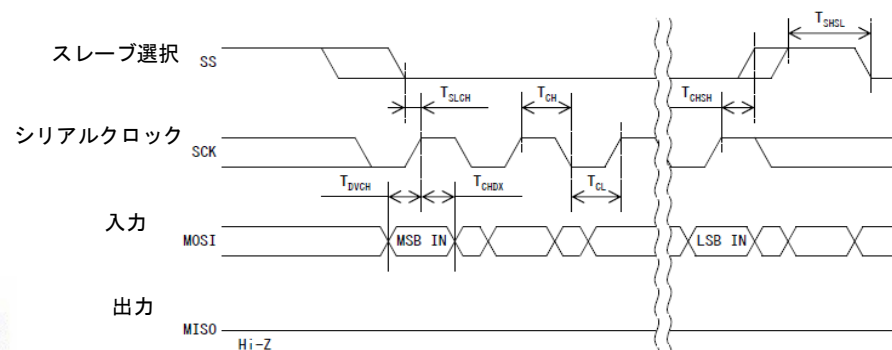
Master

《リードサイクル》



Master * SlaveはSCKの立下りでデータを出力しMasterはSCKの立上りでデータを読み込む

《ライトサイクル》



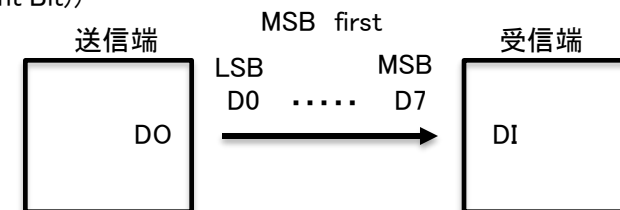
* MasterはSCKの立下りでデータを出力しSlaveはSCKの立上りでデータを読み込む

4. PCL6125アクセス関数 accessPCL.sc

・FT232HL、シリアルバス_SPI 送受信(Data In/Out)設定

3.2 Data Shifting Command Overview

Data Bit	Definition	
Bit 0	-ve TCK/SK on write	SPIのシリアルクロック_書き込みタイミング 1:-ve(立下り) 0:+ve(立上り)
Bit 1	bit mode = 1 else byte mode	モード 1:bit 0:byte
Bit 2	-ve TCK/SK on read	SPIのシリアルクロック_読み込みタイミング 1:-ve(立下り) 0:+ve(立上り)
Bit 3	LSB first = 1 else MSB first	1:LSB 0:MSM (MSB/LSB (Most Significant Bit/Least Significant Bit))
Bit 4	Do write TDI/DO	1:Data Out
Bit 5	Do read TDO/DI	1:Data In
Bit 6	Do write TMS/CS	1:CS Out
Bit 7	0	未使用



3.3.2 Clock Data Bytes Out on -ve Clock Edge MSB First (no read)

0x11
LengthL,
LengthH,
Byte1
..
Byte65536 (max)

```
FtBuff[FtIndex++] = 0x11; //SPI(Clock Data Bytes Out on -ve Clock Edge MSB First (no Read))  
FtBuff[FtIndex++] = 0x05; //Length L 6byte  
FtBuff[FtIndex++] = 0x00; //Length H
```

0x11:
Bit0 1:書き込みタイミング-ve(立下り)
Bit1 0:byteモード
Bit2 0:読込タイミング+ve(立上り)
Bit3 0:MSB first
Bit4 1:Data Out
0x05:Length L 6byte
0x00:Length H

3.3.9 Clock Data Bytes In and Out MSB First

Out on negative edge, In on negative edge

0x35,
LengthL,
LengthH,
Byte1
..
Byte65536 (max)

```
FtBuff[FtIndex++] = 0x35; //SPI(Out on negative edge, in on negative edge)  
FtBuff[FtIndex++] = 0x02; //Length L 3byte  
FtBuff[FtIndex++] = 0x00; //Length H
```

0x35
Bit0 1:書き込みタイミング-ve(立下り)
Bit1 0:byteモード
Bit2 1:読込タイミング-ve(立下り)
Bit3 0:MSB first
Bit4 1:Data Out
Bit5 1:Data In
0x02: Length L 3byte
0x00: Length H

4. PCL6125アクセス関数 accessPCL.sc

・PCL6125、シリアルバスアクセス方法（PCL61x5取扱説明書）

①書き込みフォーマット(MOSI: Master Out Slave In)

- ・DATは各軸4byte、X軸からU軸・下位byteから上位byteの順
- ・4byteに不足する場合は0を付加する、12345hの場合3byte不足 45h、23h、01h、00h

SEL	COM	DATx				DATy				DATz				DATu			
S7-S0	C7-C0	D31x-D0x				D31y-D0y				D31z-D0z				D31u-D0u			
		D7x-D0x	D15x-D8x	D23x-D16x	D31x-D24x	D7y-D0y	D15y-D8y	D23y-D16y	D31y-D24y	D7z-D0z	D15z-D8z	D23z-D16z	D31z-D24z	D7u-D0u	D15u-D8u	D23u-D16u	D31u-D24u

②読み出しフォーマット(MISO: Master In Slave Out)

- ・4byte単位、X軸からU軸の順、下位byteから上位byteの順、MSBからLSBの順に並ぶ

DATx				DATy				DATz				DATu			
D31x-D0x				D31y-D0y				D31z-D0z				D31u-D0u			
D7x-D0x	D15x-D8x	D23x-D16x	D31x-D24x	D7y-D0y	D15y-D8y	D23y-D16y	D31y-D24y	D7z-D0z	D15z-D8z	D23z-D16z	D31z-D24z	D7u-D0u	D15u-D8u	D23u-D16u	D31u-D24u

A. 軸選択(SEL)

SEL							
S7	S6	S5	S4	S3	S2	S1	S0
デバイス選択		タイプ選択コード		軸選択コード			

a. デバイス選択

- ・シリアルバスのデバイス選択端子(DS1,DS0)

S7	S6	DS1,DS0端子
0	0	DS1:L, DS0:L
0	1	DS1:L, DS0:H
1	0	DS1:H, DS0:L
1	1	DS1:H, DS0:H

b. タイプ選択コード

- ・通信フォーマット

S5	S4	内容
0	0	コマンド書き込み(レジスタ書き込み・読み出しを含む)
0	1	メインステータス読み出し
1	0	汎用出力ポート書き込み
1	1	汎用入出力ポート&サブステータス読み出し

c. 軸選択コード

PCL6125の場合、0001b:1軸目、0010b:2軸目、0011b:1, 2軸両方

S3	S2	S1	S0	LSI型式
SELu	SELz	SELy	SELx	PCL6145、4軸制御
0	0	SELy	SELx	PCL6125、2軸制御
0	0	0	SELx	PCL6115、1軸制御

B. コマンド(COM)

COM							
C7	C6	C5	C4	C3	C2	C1	C0
コマンドコード							

C. データ(DAT)

DATn			
D7n-D0n	D15n-D8n	D23n-D16n	D24n-D31n
各種データ			

- ・メインステータス読出し 各軸2byte
2byte単位、X軸からU軸の順、下位byteから上位byteの順、MSBからLSBの順に並ぶ
- ・汎用出力ポート書き込み 各軸1byte
X軸からU軸の順に並べる、8bit未満には0を付加する、1100111b→0110001111b
- ・汎用入力読出し 1byte + サブステータス読出し 1byte
2byte単位、X軸からU軸の順、下位byteから上位byteの順、MSBからLSBの順に並ぶ

4. PCL6125アクセス関数 accessPCL.sc

◆PCL6125アクセス関数 accessPCL.sc 一覧

関数名	内容
Read_STATUS	ステータスの読出し
Read_REG	レジスタの読出し
Write_REG	レジスタへの書き込み
Write_COM	動作コマンドの書き込み
SendUsb	PCL6125-EBボードへの送信
GetUsb	PCL6125-EBボードからのデータ受信

4. PCL6125アクセス関数 accessPCL.sc

•Read_STATUS ステータスの読出し

```
16 //////////////////////////////////////
17 /// <summary>
18 /// ステータス、サブステータスの読出し
19 /// Read status, sub status
20 /// subS : 0=status 1=sub status
21 /// </summary>
22 /// <param name="description"></param>
23 /// <returns></returns>
24 public void Read_STATUS(ref byte[] FtBuff, ref int FtIndex, int subS)
25 {
26     //CS=Low
27     FtBuff[FtIndex++] = 0x80;
28     FtBuff[FtIndex++] = 0x00;
29     FtBuff[FtIndex++] = 0x0B;
30     //
31     FtBuff[FtIndex++] = 0x35; // SPI (Out on negative edge, in on negative edge)
32     FtBuff[FtIndex++] = 0x04; // Length L 5byte
33     FtBuff[FtIndex++] = 0x00; // Length H
34     if(subS == 0)
35     {
36         FtBuff[FtIndex++] = 0x13; // command status read X,Y Axis
37     }
38     else
39     {
40         FtBuff[FtIndex++] = 0x33; // command sub status read X,Y Axis
41     }
42     FtBuff[FtIndex++] = 0x00; // X-axis dummy
43     FtBuff[FtIndex++] = 0x00; // Y-axis dummy
44     FtBuff[FtIndex++] = 0x00; // X-axis dummy
45     FtBuff[FtIndex++] = 0x00; // Y-axis dummy
46     //CS=High
47     FtBuff[FtIndex++] = 0x80;
48     FtBuff[FtIndex++] = 0x0B;
49     FtBuff[FtIndex++] = 0x0B;
50 }
```

FT232HL、SPIシリアルバス CS信号⇒Low

FT232HL、PCL6125へのデータ送受信 5byte

SEL:0x13 メインステータスの読出し、X,Y両軸を選択

SEL:0x33 サブステータス&汎用入出力ポート読出し
X,Y両軸を選択

PCL6125_シリアルバスアクセス
5byte

サブステータス(SSTS) 2byte

サブステータス(SSTS) 2byte

FT232HL、SPIシリアルバス CS信号⇒High

4. PCL6125アクセス関数 accessPCL.sc

•Read_REG レジスタの読出し

```
52 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
53 /// <summary>
54 /// レジスタの読出し
55 /// Read registers
56 /// comm : レジスタリードコマンド
57 /// Register read command
58 /// </summary>
59 /// <param name="description"></param>
60 /// <returns></returns>
61 public void Read_REG(ref byte[] FtBuff, ref int FtIndex, byte comm)
62 {
63     //CS=Low
64     FtBuff[FtIndex++] = 0x80;
65     FtBuff[FtIndex++] = 0x00;
66     FtBuff[FtIndex++] = 0x0B;
67     //
68     FtBuff[FtIndex++] = 0x35; // SPI (Out on negative edge, in on negative edge)
69     FtBuff[FtIndex++] = 0x09; // Length L 10byte
70     FtBuff[FtIndex++] = 0x00; // Length H
71     FtBuff[FtIndex++] = 0x03; // Command write SEL:0x03 コマンドの書込み、X,Y両軸を選択
72     FtBuff[FtIndex++] = comm; // COM:comm コマンド
73     FtBuff[FtIndex++] = 0x00; // X-axis dummy } DATx:D31-D0 4byte
74     FtBuff[FtIndex++] = 0x00;
75     FtBuff[FtIndex++] = 0x00;
76     FtBuff[FtIndex++] = 0x00;
77     FtBuff[FtIndex++] = 0x00; // Y-axis dummy } DATy:D31-D0 4byte
78     FtBuff[FtIndex++] = 0x00;
79     FtBuff[FtIndex++] = 0x00;
80     FtBuff[FtIndex++] = 0x00;
81     //CS=High
82     FtBuff[FtIndex++] = 0x80;
83     FtBuff[FtIndex++] = 0x08;
84     FtBuff[FtIndex++] = 0x0B;
85 }
```

FT232HL、SPIシリアルバス CS信号⇒Low

FT232HL、PCL6125へのデータ送受信 10byte

SEL:0x03 コマンドの書込み、X,Y両軸を選択
COM:comm コマンド

DATx:D31-D0 4byte

DATy:D31-D0 4byte

PCL6125_シリアルバスアクセス
10byte

FT232HL、SPIシリアルバス CS信号⇒High

4. PCL6125アクセス関数 accessPCL.sc

•Write_REG レジスタの書き込み

```
87 ///////////////////////////////////////////////////////////////////
88 /// <summary>
89 /// レジスタの書き込み
90 /// Register write
91 ///   RegD : 書き込みデータ
92 ///         Write data
93 ///   Jsc  : 軸選択コマンド
94 ///         Axis selection command
95 ///   comm : レジスタライトコマンド
96 ///         Register write command
97 /// </summary>
98 /// <param name="description"></param>
99 /// <returns></returns>
100 public void Write_REG(ref byte[] FtBuff, ref int FtIndex, uint RegD, byte Jsc, byte comm)
101 {
102     ///////////////////////////////////////////////////////////////////
103     var bytes = new TBytes();
104
105     bytes.Word = RegD;
106
107     //CS=Low
108     FtBuff[FtIndex++] = 0x80;
109     FtBuff[FtIndex++] = 0x00;
110     FtBuff[FtIndex++] = 0x0B;
111     //
112     FtBuff[FtIndex++] = 0x11; // SPI (Clock Data Bytes Out on -ve Clock Edge MSB First (no Read))
113     FtBuff[FtIndex++] = 0x05; // Length L 6byte
114     FtBuff[FtIndex++] = 0x00; // Length H
115     FtBuff[FtIndex++] = Jsc;
116     FtBuff[FtIndex++] = comm;
117     FtBuff[FtIndex++] = bytes.Byte0; // Data 7-0
118     FtBuff[FtIndex++] = bytes.Byte1; // Data 15-8
119     FtBuff[FtIndex++] = bytes.Byte2; // Data 23-16
120     FtBuff[FtIndex++] = bytes.Byte3; // Data 31-24
121     //CS=High
122     FtBuff[FtIndex++] = 0x80;
123     FtBuff[FtIndex++] = 0x0B;
124     FtBuff[FtIndex++] = 0x0B;
125 }
```

FT232HL、SPIシリアルバス CS信号⇒Low

FT232HL、PCL6125へのデータ送信 6byte

軸選択、0x01:1軸、0x02:Y軸、0x03:X,Y両軸
COM:comm コマンド

DAT:D31-D0 4byte

PCL6125_シリアルバスアクセス
6byte

FT232HL、SPIシリアルバス CS信号⇒High

4. PCL6125アクセス関数 `accessPCL.sc`

・Write_COM 動作コマンドの書込み

```

127 //////////////////////////////////////
128 /// <summary>
129 /// コマンドの書込み
130 /// Write command
131 /// Jsc : 軸選択コマンド
132 ///      Axis selection command
133 /// comm : コマンド
134 ///      Command
135 /// </summary>
136 /// <param name="description"></param>
137 /// <returns></returns>
138 public void Write_COM(ref byte[] FtBuff, ref int FtIndex, byte Jsc, byte comm)
139 {
140     //////////////////////////////////////
141     //CS=Low
142     FtBuff[FtIndex++] = 0x80;
143     FtBuff[FtIndex++] = 0x00;
144     FtBuff[FtIndex++] = 0x0B;
145     //
146     FtBuff[FtIndex++] = 0x11; // SPI(Clock Data Bytes Out on -ve Clock Edge MSB)
147     FtBuff[FtIndex++] = 0x01; // Length L 2byte
148     FtBuff[FtIndex++] = 0x00; // Length H
149     FtBuff[FtIndex++] = Jsc;
150     FtBuff[FtIndex++] = comm;
151     //CS=High
152     FtBuff[FtIndex++] = 0x80;
153     FtBuff[FtIndex++] = 0x0B;
154     FtBuff[FtIndex++] = 0x0B;
155 }

```

FT232HL、SPIシリアルバス CS信号

FT232HL、PCL6125へのデータ送信

軸選択、0x01:1軸、0x02:Y軸、0x03:X軸
COM:comm コマンド

FT232HL、SPIシリアルバス CS信号

【例】

次の順番でレジスタへのデータ、動作コマンドを格納し、実行します。

```

FtIndex=0;
hAPCL_Write_REG(ref FtBuff, ref FtIndex, 0x00000001, 0x01, 0x81); // write PRFL (X-Axis)
hAPCL_Write_REG(ref FtBuff, ref FtIndex, 0x00000400, 0x01, 0x82); // write PRFH (X-Axis)
hAPCL_Write_REG(ref FtBuff, ref FtIndex, 0x00002588, 0x01, 0x83); // write PRUR (X-Axis)
hAPCL_Write_REG(ref FtBuff, ref FtIndex, 0xFFFF700, 0x01, 0x80); // write PRMV (X-Axis)
//
hAPCL_Write_COM(ref FtBuff, ref FtIndex, 0x01, 0x53); // write High speed start 2 (X-Axis)
//
ftStatus = hAPCL.SendUsb(ref FtBuff, ref FtIndex);

```

FT232HL、SPIシリアルバス CS信号⇒Low

FT232HL、PCL6125へのデータ送信 2byte

data Bytes Out on -ve Clock Edge MSB First (no Read))

te

軸選択、0x01:1軸、0x02:Y軸、0x03:X,Y両軸
COM:comm コマンド

PCL6125_シリアルバスアクセス
2byte

FT232HL、SPIシリアルバス CS信号⇒High

4. PCL6125アクセス関数 accessPCL.sc

•SendUsb PCL6125-EBボードへの送信

```
157 //////////////////////////////////////
158 /// <summary>
159 /// 積み込んだデータをUSB送信する
160 /// Sending loaded data from USB
161 /// </summary>
162 /// <param name="description"></param>
163 /// <returns></returns>
164 public FTDI.FT_STATUS SendUsb(ref byte[] FtBuff, ref int FtIndex)
165 {
166     //////////////////////////////////////
167     FTDI.FT_STATUS ftStatus = FTDI.FT_STATUS.FT_OK;
168
169     if (FtIndex > 0)
170     {
171         ftStatus = hFTDI.LSI_Write(FtBuff, FtIndex);
172         FtIndex = 0;
173     }
174     //
175     return (ftStatus);
176 }
```

LSI_Write:FTDI Simple Write関数

バッファに格納されているコマンド類を、PCL6125-EBに送信

送信後、コマンド類はPCL6125-EBボード側で実行される

FtBuff

「Read_STATUS」、「Read_REG」、「Write_REG」、「Write_COM」等の関数で
コマンド群を格納したバッファを指定

FtIndex

バッファの使用数を指定、送信後この変数はクリアされる

4. PCL6125アクセス関数 accessPCL.sc

・GetUsb PCL6125-EBボードからのデータ受信

```
178 ////////////////////////////////////////  
179 /// <summary>  
180 /// USB側のデータを送信させる  
181 /// Transmit the USB side data  
182 /// </summary>  
183 /// <param name="description"></param>  
184 /// <returns></returns>  
185 public FTDI.FT_STATUS GetUsb(ref byte[] FtBuff)  
186 {  
187     FTDI.FT_STATUS ftStatus = FTDI.FT_STATUS.FT_OK;  
188     uint numBytesRead = 0;  
189  
190     ////////////////////////////////////////  
191     ftStatus = hFTDI.LSI_Read(ref FtBuff, ref numBytesRead);  
192     //  
193     return (ftStatus);  
194 }
```

【例】

次の順番でコマンドを格納し、実行したと想定します。

```
FtIndex=0;  
Read_STATUS(ref FtBuff, ref FtIndex, 0); // メインステータスリード  
Read_REG(ref FtBuff, ref FtIndex, 0xE3); // RCUN1 レジスタリード  
Read_STATUS(ref FtBuff, ref FtIndex, 1); // サブステータスリード  
//  
SendUsb(ref FtBuff, ref FtIndex);  
GetUsb(ref FtBuff);
```

LSI_Read:FTDI Simple Read関数

バッファに格納されているコマンド類を、PCL6125-EBに送信
送信後、コマンド類はPCL6125-EBボード側で実行される

FtBuff

実行した読出しコマンドの実行結果の順番で、読みだされたデータが
格納される

受信後のバッファは次のようになります。

FtBuff		
0	ステータス bit7~bit0	// X 軸メインステータスリード
1	ステータス bit15~bit8	
2	ステータス bit7~bit0	// Y 軸メインステータスリード
3	ステータス bit15~bit8	
4	RCUN1 レジスタ bit7~bit0	// X 軸 RCUN1 レジスタリード
5	RCUN1 レジスタ bit15~bit8	
6	RCUN1 レジスタ bit23~bit16	
7	RCUN1 レジスタ bit31~bit24	
8	RCUN1 レジスタ bit7~bit0	// Y 軸 RCUN1 レジスタリード
9	RCUN1 レジスタ bit15~bit8	
10	RCUN1 レジスタ bit23~bit16	
11	RCUN1 レジスタ bit31~bit24	
12	汎用ポートの状態	// X 軸サブステータスリード
13	サブステータス	
14	汎用ポートの状態	// Y 軸サブステータスリード
15	サブステータス	

4. PCL6125アクセス関数 accessPCL.sc

【参考】 パラレルバスアクセス方法（CPU(8086系列)とPCL6125の接続）（PCL61x5取扱説明書）

・アドレスマップ

アドレス					読出し		書込み		軸
A4	A3	A2	A1	hex	記号	内容	記号	内容	
0	0	0	0	00h	MSTSW	メインステータス(Bit 15～0)	COMW	軸選択、コマンド	X軸
0	0	0	1	02h	SSTSW	サブステータス、汎用出力ポート	OTPW	汎用出力ポートの状態変更	
0	0	1	0	04h	BUFW0	入出力バッファ(Bit 15～0)	BUFW0	入出力バッファ(Bit 15～0)	
0	0	1	1	06h	BUFW1	入出力バッファ(Bit 31～16)	BUFW1	入出力バッファ(Bit 31～16)	
0	1	0	0	08h	MSTSW	メインステータス(Bit 15～0)	COMW	軸選択、コマンド	Y軸
0	1	0	1	0Ah	SSTSW	サブステータス、汎用出力ポート	OTPW	汎用出力ポートの状態変更	
0	1	1	0	0Ch	BUFW0	入出力バッファ(Bit 15～0)	BUFW0	入出力バッファ(Bit 15～0)	
0	1	1	1	0Dh	BUFW1	入出力バッファ(Bit 31～16)	BUFW1	入出力バッファ(Bit 31～16)	
1	0	0	0	10h	MSTSW	メインステータス(Bit 15～0)	COMW	軸選択、コマンド	Z軸
1	0	0	1	12h	SSTSW	サブステータス、汎用出力ポート	OTPW	汎用出力ポートの状態変更	
1	0	1	0	14h	BUFW0	入出力バッファ(Bit 15～0)	BUFW0	入出力バッファ(Bit 15～0)	
1	0	1	1	16h	BUFW1	入出力バッファ(Bit 31～16)	BUFW1	入出力バッファ(Bit 31～16)	
1	1	0	0	18h	MSTSW	メインステータス(Bit 15～0)	COMW	軸選択、コマンド	U軸
1	1	0	1	1Ah	SSTSW	サブステータス、汎用出力ポート	OTPW	汎用出力ポートの状態変更	
1	1	1	0	1Ch	BUFW0	入出力バッファ(Bit 15～0)	BUFW0	入出力バッファ(Bit 15～0)	
1	1	1	1	1Dh	BUFW1	入出力バッファ(Bit 31～16)	BUFW1	入出力バッファ(Bit 31～16)	

・コマンド(COM) 16bit

注、PCL6115はA4,A3端子はない、PCL6125はA4端子はない

- ①動作させたい軸、SELx-uに”1”を入れ指定、複数軸に”1”を入れると同時に指定が可能
- ②SELx-uすべてに”0”を入れた場合、A4,A3で指定した軸が有効（注:PCL6115は1軸なのでSELx-uは無視する）

word byte bit	COMW															
	COMB1								COMB0							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	SELu	SELz	SELy	SELx	COM							

・入出力バッファ(BUF) 32bit

書込み: COMWにレジスタ書込みコマンドを書込むと入出力バッファのデータがレジスタにコピーされる
(参考、BUFW1,0のデータの書込み順序はない、書き込んだデータは読出し可能)

読出し: COMWにレジスタ読出しコマンドを書込むとレジスタから入出力バッファにコピーされる

word byte bit	BUFW0															
	BUFB1								BUFB0							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
word byte bit	BUFW1															
	BUFB3								BUFB2							
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

4. PCL6125アクセス関数 accessPCL.sc

軸選択とアクセス方法

・次の2種類のどちらでも可能

1アドレスのコマンドエリアを使う方法と各軸に割り振られたコマンドエリアを使う方法

①の方法

・アドレス00h(A4-A1:0000b)に02DChを書込む

上位02h ⇒ 軸選択 Y軸

下位DCh ⇒ コマンド DCh 環境設定1レジスタ読出し

・次から環境設定1レジスタの値を読出し

アドレス0Ch(A4-A1:0110b) 環境設定1レジスタ(Bit 15～ 0)

アドレス0Dh(A4-A1:0111b) 環境設定1レジスタ(Bit 31～16)

②の方法

・アドレス08h(A4-A1:0100b)に00DChを書込む

上位00h ⇒ 軸選択 A4,A3が優先されY軸

下位DCh ⇒ コマンド DCh 環境設定1レジスタ読出し

・次から環境設定1レジスタの値を読出し

アドレス0Ch(A4-A1:0110b) 環境設定1レジスタ(Bit 15～ 0)

アドレス0Dh(A4-A1:0111b) 環境設定1レジスタ(Bit 31～16)

①

A4～A1	アドレス記号	内容
0000	COMW	軸選択、コマンド
0010	BUF0_X	X軸用入出力バッファ(Bit 15～ 0)
0011	BUF1_X	X軸用入出力バッファ(Bit 31～16)
0110	BUF0_Y	Y軸用入出力バッファ(Bit 15～ 0)
0111	BUF1_Y	Y軸用入出力バッファ(Bit 31～16)
1010	BUF0_Z	Z軸用入出力バッファ(Bit 15～ 0)
1011	BUF1_Z	Z軸用入出力バッファ(Bit 31～16)
1110	BUF0_U	U軸用入出力バッファ(Bit 15～ 0)
1111	BUF1_U	U軸用入出力バッファ(Bit 31～16)

②

A4～A1	アドレス記号	内容
0000	COMW_X	X軸用コマンド
0010	BUF0_X	X軸用入出力バッファ(Bit 15～ 0)
0011	BUF1_X	X軸用入出力バッファ(Bit 31～16)
0100	COMW_Y	Y軸用コマンド
0110	BUF0_Y	Y軸用入出力バッファ(Bit 15～ 0)
0111	BUF1_Y	Y軸用入出力バッファ(Bit 31～16)
1000	COMW_Z	Z軸用コマンド
1010	BUF0_Z	Z軸用入出力バッファ(Bit 15～ 0)
1011	BUF1_Z	Z軸用入出力バッファ(Bit 31～16)
1100	COMW_U	U軸用コマンド
1110	BUF0_U	U軸用入出力バッファ(Bit 15～ 0)
1111	BUF1_U	U軸用入出力バッファ(Bit 31～16)

4. PCL6125アクセス関数 accessPCL.sc

・メインステータス(MSTS) 16bit

読出し: 各軸のアドレスのデータを読出す

word	MSTSW															
byte	MSTSB1								MSTSB0							
bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

・汎用出力ポート(OPT) 8bit

書込み: 各軸のアドレスにデータを書込む

word	OPTW															
byte	-								OPTB							
bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0								

・サブステータス(SSTS) 8bit と汎用入力ポート(IOP) 8bit

読出し: 各軸のアドレスデータを読出す、Wordで読んだ場合は下位1byteは汎用入力、上位1byteはサブステータス

word	SSTSW															
byte	SSTSB								IOPB							
bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

