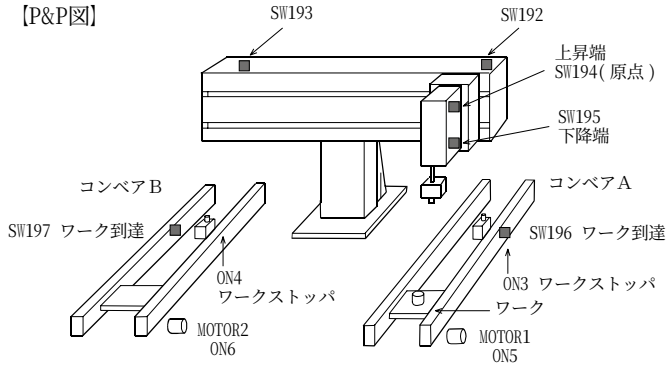
第３章　プログラム例

I/O制御・マルチタスク

MPC の構成と装置概要

図のようなP&Pのプログラムを考えます。このP&Pはコンベアー Ａでパレットに乗って搬送されているワークをストッパーで止め、エアーシリンダーで構成されたP&Pでコンベアー Ｂのパレットへ乗せかえるというものです。入力はフォトマイクロセンサーまたはリードスイッチ、出力モーターはＳＳＲを介してON/OFFを行いそれ以外はソレノイドバルブを直接駆動するという設定です。この例でのMPCのハード構成はMPC-684、MIP-048、MOP-048、RACK-N6を各1台ずつ、アドレス設定は出荷時のまま使用します。

【P&P図】



I/Oマップ

入力 (MIP-048)　　出荷時の設定ではポート番号は192から始まります。

ポート	接続機器	名　称	機　能
１９２	シリンダーセンサー	ｋｏｔａｎ	Ｐ＆Ｐ後退端
１９３	シリンダーセンサー	ｚｅｎｔａｎ	Ｐ＆Ｐ前進端
１９４	シリンダーセンサー	ｕｅ	Ｐ＆Ｐ上昇端
１９５	シリンダーセンサー	ｓｈｉｔａ	Ｐ＆Ｐ下降端
１９６	フォトマイクロセンサー	ｗｏｒｋ１	コンベアーＡワーク到達
１９７	フォトマイクロセンサー	ｗｏｒｋ２	コンベアーＢワーク到達

出力 (MOP-048)　　出荷時の設定ではポート番号は0から始まります。

ポート	接続機器	名　称	機　能
0	ソレノイドバルブ	ｚｅｎｓｈｉｎ	Ｐ＆Ｐ前進
1	ソレノイドバルブ	ｋａｋｏｕ	Ｐ＆Ｐ下降
2	ソレノイドバルブ	ｃｈａｃｋ	メカチャック閉
3	ソレノイドバルブ	ｓｔｏｐｐｅｒ１	コンベアーＡストッパー
4	ソレノイドバルブ	ｓｔｏｐｐｅｒ２	コンベアーＢストッパー
5	ＳＳＲ	ｍｏｔｏｒ１	コンベアーＡモーター
6	ＳＳＲ	ｍｏｔｏｒ２	コンベアーＢモーター

メモリーI/O　　メモリーI/Oは-1から-8192です。

ポート	接続機能	名　称	機　能
－１	メモリーＩ／Ｏ	ａ＿ｒｅａｄｙ	コンベアーＡＲＥＡＤＹ
－２	メモリーＩ／Ｏ	ｂ＿ｒｅａｄｙ	コンベアーＢＲＥＡＤＹ

この装置の動作は単純で A から B への移し替えの仕事は1つのシーケンスとしてプログラム出来るかも知れません。しかし、実際の多くの装置はとても複雑で、タクトタイムなどを考えると適当なユニット単位に分割してそれぞれが独立して動作するように制御するのが合理的です。例えばこの装置の場合コンベアー A の部分、P&Pの部分、コンベアー B の部分という3つブロックに分けてプログラムします。このブロックをタスクと言い、MPC-684では複数のタスクを同時に実行するのでマルチタスクと言います。しかしそれぞれのタスクが勝手に動作していたのでは装置としての目的を達成することは出来ません。そこでタスク同士の連絡を行う必要がありこれをインターロックと言います。インターロックは、メモリーI/Oもしくは変数で行います。メモリーI/Oとは通常装置間のインターロックを物理的なI/Oで行うようにタスク間のインターロックを行うための内部的なI/Oです。メモリーI/Oは物理的なI/Oと同様に1つのビットで1(ON)または0(OFF)の2つの状態を表します。そしてMPC-684ではON/OFF/SWなどのコマンドに引き数を負の整数として与える事によりメモリーI/Oとして、物理的なI/Oと同様に扱う事ができます。1つのメモリーI/OではON/OFF のどちらしか無いのに対し、変数を用いると複数の情報を伝えることができます。MPC-684 の変数は4byte整数ですから、1つの変数で約±20億の状態を表すことができます。

I/O のシンボル化

プログラム中の入出力ポートのパラメーターとして I/O のポートナンバーを整数でそのまま記述する事ができますが、シンボルとして扱うと見た目にも分かりやすいプログラムになります。次のようにCONSTコマンドで定数をシンボル化します。

```
CONST  sol1 0
CONST  sw1  192
```

これでON 0とON sol1,WAIT SW(192)=0はWAIT SW(sw1)=0と同義になります。
(P&P図) の装置のI/O定義

```
10 'I/O定義
20 '入力
30 CONST kotan 192
40 CONST zentan 193
50 CONST ue 194
60 CONST shita 195
70 CONST work1 196
80 CONST work2 197
90 '出力
100 CONST zenshin 0
110 CONST kakou 1
120 CONST chack 2
130 CONST stopper1 3
140 CONST stopper2 4
150 CONST motor1 5
160 CONST motor2 6
170 'メモリー I/O
180 CONST a_ready -1
190 CONST b_ready -2
200 'I/O初期化
210 SETIO
```

タスクの生成

それぞれのユニットを1～31のどのタスクで実行するかを宣言します。0はメインのタスクで、プログラムが実行されると最初にこのタスクが動きます。この例ではP&Pはタスク0で、コンベアー A はタスク1、コンベアー B はタスク2で動かします。

```
220 'タスク実行
230 FORK 1 *conv_a        ’タスク1でコンベアー Aを実行
240 FORK 2 *conv_b        ’タスク2でコンベアー Bを実行
```

P&P のタスク

P&P はコンベアー A からの READY を待ちワークをピックアップしコンベアー B 側に搬送します。コンベアー B の準備が出来たところでワークを置きます。このタスクはメインタスク（タスク0）で実行します。

```
1000 *PANDP
1010 '
1020 DO
1030 WAIT SW(a_ready)==1   ’コンベアＡREADY待ち
1040 '
1050 ON kakou              ’ワークピックアップ
1060 WAIT SW(shita)==1
1070 ON chack
1080 TIME 500
1090 OFF kakou
1100 WAIT SW(ue)==1
1110 '
1120 OFF a_ready           ’ピックアップ完了－＞コンベアＡ
1130 '
1140 ON zenshin            ’P&P前進
1150 WAIT SW(zentan)==1
1160 '
1170 WAIT SW(b_ready)==1   ’コンベアー Ｂ準備完了待ち
1180 '
1190 ON kakou              ’ワークプレース
1200 WAIT SW(shita)==1
1210 OFF chack
1220 TIME 300
1230 OFF kakou
1240 WAIT SW(ue)==1
1250 '
1260 OFF b_ready           ’ワークプレース完了－＞コンベアー
1270 '
1280 OFF zenshin           ’P&P後退
1290 WAIT SW(zentan)==1
1300 '
1310 LOOP
```

コンベアー A のタスク

コンベアー A はストッパーを上げmotor1を駆動してパレットが到達するのを待ちます。パレットが到達するとP&PにREADYを出し、P&Pはワークを持ち上げるとREADYをOFFします。

```
2000 *conv_a
2010 ON motor1
2020 DO
2030 ON stopper1
2040 WAIT SW(work)==1
2050 TIME 500
2060 OFF motor1
2070 '
2080 ON a_ready            ’ワークが来たヨ
2090 WAIT SW(a_ready)==0   ’P&P動作完了待ち
2100 '
2110 OFF stopper1
2120 ON motor1
2130 WAIT SW(work1)==0
2140 TIME 500
2150 LOOP
```

コンベアー B のタスク

コンベアー Bはモーターを回しストッパーを上げ空パレットが到達するのを待ち、P&PへREADYを出しP&Pはワークを置くとREADYをOFFします。

```
3000 *conv_b
3010 ON motor2
3020 DO
3030 ON stopper2
3040 WAIT SW(work2)==1
3050 TIME 500
3060 OFF motor2
3070 '
3080 ON b_ready            ’パレット準備完了
3090 WAIT SW(b_ready)==1   ’ワークを置いたヨ
3100 '
3110 ON motor2
3120 OFF stopper2
3130 WAIT SW(work2)==0
3140 TIME 500
3150 LOOP
```

装置全体のプログラム

前記プログラムを入力してRENUMすると次のLISTになります。

```
10   'I/O定義
20   '入力
30    CONST kotan 192
40    CONST zentan 193
50    CONST ue 194
60    CONST shita 195
70    CONST work1 196
80    CONST work2 197
90   '出力
100   CONST zenshin 0
110   CONST kakou 1
120   CONST chack 2
130   CONST stopper1 3
140   CONST stopper2 4
150   CONST motor1 5
160   CONST motor2 6
170  'メモリー I/O
180   CONST a_ready -1
190   CONST b_ready -2
200  'I/O初期化
210   SETIO
220  'タスク実行
230   FORK 1 *conv_a                ’タスク1でコンベアー Aを実行
240   FORK 2 *conv_b                ’タスク2でコンベアー Bを実行
250  *PANDP
260  '
270   DO
280     WAIT SW(a_ready)==1         ’コンベアAREADY待ち
290  '
300     ON kakou                    ’ワークピックアップ
310     WAIT SW(shita)==1
320      ON chack
330      TIME 500
340      OFF kakou
350      WAIT SW(ue)==1
360  '
370      OFF a_ready                ’ピックアップ完了－＞コンベアA
380  '
390      ON zenshin                 ’P&P前進
```

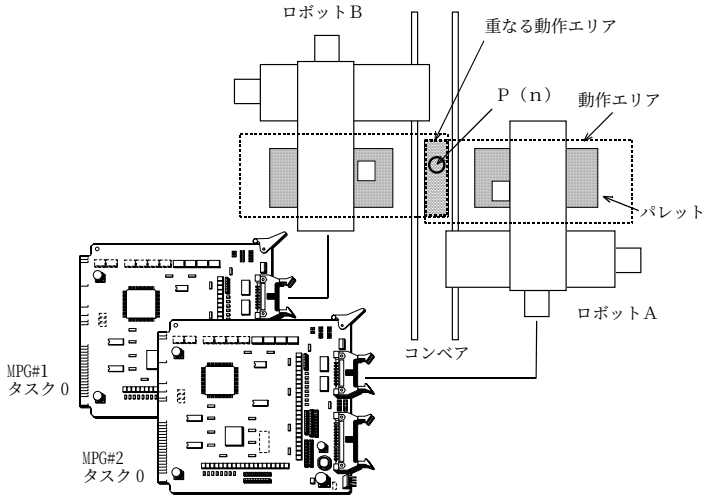
```
400       WAIT SW(zentan)==1
410   '
420     WAIT SW(b_ready)==1             ’コンベアー Ｂ準備完了待ち
430   '
440     ON kakou                        ’ワークプレース
450     WAIT SW(shita)==1
460     OFF chack
470     TIME 300
480     OFF kakou
490     WAIT sw(ue)==1
500   '
510     OFF b_ready                     ’ワークプレース完了－＞コンベアー Ｂ
520   '
530     OFF zenshin                     ’P&P後退
540     WAIT SW(zentan)==1
550   '
560     LOOP
570   *conv_a
580     ON motor1
590     DO
600     ON stopper1
610     WAIT SW(work1)==1
620     TIME 500
630     OFF motor1
640   '
650     ON a_ready                      ’ワークが来たヨ
660     WAIT SW(a_ready)==0             ’P&P動作完了待ち
670   '
680     OFF stopper1
690     ON motor1
700     WAIT SW(work1)==0
710     TIME 500
720     LOOP
730   *conv_b
740     ON motor2
750     DO
760       ON stopper2
770       WAIT SW(work2)==1
780       TIME 500
790       OFF motor2
800   '
810     ON b_ready                      ’パレット準備完了
820     WAIT SW(b_ready)==0             ’ワークを置いたヨ
830   '
840     ON motor2
850     OFF stopper2
860     WAIT SW(work2)==0
870     TIME 500
880     LOOP
```

タスクの制御

タスクの交通整理（セマフォ）

タスク間のインターロックが必要なケースとして先の例でのタスクの同期と、もう1つ、一度に1つのタスクしか使用できない共通資源（クリティカルソース）に対し排他処理を行う場合があります。例えば、1つのRS-232 ポートやリエントラント出来ないサブルーチンを複数のタスクで使用したり、重なる動作エリアを持つ2台のロボット（図）の干渉防止などがあります。本来、前者のような一人の聞き手に対して一度に何人もの人が同時に話しかけるようなプログラムは避けるべきです。聖徳太子のように器用に聞き分けることはできないのです。1つのデバイスは1つのタスクで管理することが原則ですが、それができないときは複数のタスクが同時にアクセスしないようにしなければなりません。後者はタスクのアクセスする対象は異なりますが、ポイントや空間が共通資源として考えられます。このようなタスクの交通整理をセマフォと言い、それを合理的に確実に行う方法として、セマフォ関数（RSV(n),RLS(n)）があります。

【2台のロボット図】



＊コンベアー上のワークを2台のロボットのどちらかでパレットに搬送する。
片方がP(n)へアクセス中、他方は絶対にそのエリアに進入してはならない。

1. プログラムはコマンドリファレンスのRSV(n)の項のプログラムです。このプログラムは2つのタスクが勝手に走ってRS-232のCH1を同時にアクセスしてしまい出力が混乱しています。もしもこれが2台のロボットなら衝突して大きな事故になってしまいます。
2. メモリーI/OとON/OFF/SW(n)を組み合わせてテスト&セットを行うことによりタスクの衝突を防ごうとしています。しかしこの方法ではメモリーI/Oのテストとセットが別々のコマンドによって行われているため時間差が生じてタイミングによってはやはり混乱が生じる恐れがあります。
3. メモリーI/Oをセマフォ関数を使ってテスト&セットしています。RSV(-1)でメモリーI/Oの-1の状態を確認し0であればそれに1をセットする事によりセマフォの獲得をしてPRINT文を実行し、RLS(-1)でセマフォを解放します。この様にRSVは2.）のSWとONによるテスト&セットを1つ関数で行う事により確実な排他処理を実現できます。
4. RS-232へ出力する専用のタスクを設けたものです。ここでもメモリーI/Oを使ってタスクの情報交換をしています。

1.2つのタスクが勝手に動作（交通整理が行われていない）

```
                                        'もしもロボットなら
                                           ↓
10      time0=timer
20      time=timer
30    FORK 1 *aho
```

```
40   DO
50     WAIT time<>timer
60     PRINT "abcdefg" "ABCDEFGH"       ’ロボットＡがP(n)へ動作
70       time=timer
80   LOOP
90 *aho
100  DO
110    WAIT time0<>timer
120    PRINT "123456" "7890123"         ’ロボットＢがP(n)へ動作
130      time0=timer
140  LOOP
#run
123456789abcdefgABC0123                 ’ガーン衝突！エライコッチャ
DEFGH
abcdefgAB1234567890CDEFGH
123
12345678901abcdefgA23
BCDEDGH
```

(timerは予約変数です)

2.ON/OFF/SW(n)による交通整理（悪い例・交通整理できません）

```
10      time0=timer
20      time=timer
30    OFF -1
40    FORK 1 *aho
50    DO
60      WAIT time<>timer
70      WAIT SW(-1)==0
80      ON -1
90      PRINT "abcdefg" "ABCDEFGH"
100     OFF -1
110       time=timer
120   LOOP
130 *aho
140   DO
150     WAIT time0<>timer
160     WAIT SW(-1)==0
170     ON -1
180     PRINT "123456" "7890123"
190     OFF -1
200       time0=timer
210    LOOP
#run
abcdefgABCDEFGH             ’たまたまうまくいった。デモ...
1234567890123
abcdefgABCDEFGH
1234567890123
```

3.セマフォ関数を用いた交通整理（良い例）

```
10      time0=timer
20      time=timer
30    FORK 1 *aho
40    DO
50      WAIT time<>timer
60      WAIT RSV(-1)==0                 ’セマフォ確認
70      PRINT "abcdefg" "ABCDEFGH"      ’ロボットＡ動作
80        dummy=RLS(-1)                 ’セマフォ解除
90        time=timer
100   LOOP
110 *aho
120   DO
130     WAIT time0<>timer
140     WAIT RSV(-1)==0                 ’セマフォ確認
150     PRINT "123456" "7890123"        ’ロボットＢ動作
```

```
160       dummy=RLS(-1)                 ’セマフォ解除
170       time0=timer
180   LOOP
#run
abcdefgABCDEFGH                         ’ロボットＡ動作
1234567890123                           ’ロボットＢ動作
abcdefgABCDEFGH                           メデタシメデタシ
1234567890123
```

4.RS-232への出力を1本化（良い例）

```
10      time0=timer
20      time=timer
30    FORK 1 *aho
40    FORK 2 *baka
50    DO
60      IF SW(-1)==1 THEN : GOSUB *case1 : OFF -1 : END_IF
70      IF SW(-2)==1 THEN : GOSUB *case2 : OFF -2 : END IF
80    LOOP
90  *case1
100   PRINT "abcdefg" "ACDEFGH"
110   RETURN
120 *case2
130   PRINT "123456" "78901234"
140   RETURN
150 *aho
160   DO
170     WAIT time<>timer
180     ON -1
190     WAIT SW(-1)==0
200       time=timer
210   LOOP
220 *baka
230   DO
240     WAIT time0<>timer
250     ON -2
260     WAIT SW(-2)==0
270       time0=timer
280   LOOP
#run
12345678901234
abcdefgABCDEFGH
12345678901234
abcdefgABCDEFGH
```

タスクの FORK、QUIT、PAUSE

インタプリタはタスク0を親のタスクとして実行し、FORKコマンドで子タスクを実行します。子タスクには優先順位や連続性はありません。昇順、降順、中飛びは関係ありません。また、子タスク同士のFORK、PAUSE、QUITもできます。たとえば、自動モード／単動モードの切り替えを行うのに常に自動と単動の2つのタスクを走らせておく必要はありません。自動、単動それぞれルーチンを用意しておき自動モードでは単動のルーチンをQUITし、自動のルーチンをFORKします。単動モードではその逆をします。このように、1つのタスクでもFORKするプログラムを差し替えることにより複数の機能を実現する事ができます。

```
*MAIN
  DO
    WAIT SW(start)==0
    WAIT SW(start)==1       ’スタートスイッチ入力待ち
    IF SW(mode)==1 THEN
      FORK 1 *AUTO          ’modeスイッチがONならばAUTOモード
      ELSE
      FORK 1 *MANUAL
    END_IF
    WAIT SW(stop)==1        ’ストップスイッチ入力待ち
    QUIT 1                  ’タスク1をQUIT
```

```
    WAIT TASK(1)==-2
  LOOP
'*********************
*AUTO                      ’自動運転のプログラム
   DO
     JUMP P(1)
     ON 0
     JUMP P(2)
     OFF 0
   LOOP
'*********************
*MANUAL                    ’単動運転のプログラム
   DO
     JUMP P(1)
     ON 0
     WAIT SW(tando)==0
     WAIT SW(tando)==1
   LOOP
```

パルス発生について

ボードアドレスとマルチタスク

MPG-314はDSW1を0-9に設定することで、ひとつのシステムで10枚まで使用することができます。そのときのボードのアドレスはDSW番号にしたがって、&H400～&H490となります。
また、各タスク毎にPGコマンドで使用するPGを選択することができます。

```
PG &H400     /* DSW1=0のボードを使用する
PG &H420 1   /* DSW1=2のボードをタスク1に割り当てる
```

最初の確認

MPC-684はMPG-314を自動認識して初期化します。 POWER ON後LED17,18(緑)が点灯になります。
　　LED17が点灯しない->ラックに正しく挿入されていない
　　LED18が点灯しない->MPG-314にDC24Vが接続されていない
LEDの点灯を確認したらパルス発生の確認をします。
DSWの値にしたがってPGコマンドを実行し、T<Enter>します。この時J2,J3,J4はすべてコネクタをとりはずしてください。

```
#PG &h410                                  /*DSWが1の場合
#t
PG[0,410] X= 0 Y= 0 U= 0 Z= 0 dx= 500 dy= 500 du= 500 dz= 500
```

前記の表示を確認したら、x,y,z,uキーを押してみます。LED1～8のいずれかが点灯しパルス発生を確認できます。SHIFTキーを押すとCCW側にパルス発生します。
これにより簡単なインチング動作、パルス発生確認ができます。このコマンドを停止するにはQを押します。

※注意

パルスが出ない場合。
　　J2が接続され、X-LMT～Z-LMTがONになっている。
　　J2のリミット入力は、無条件に有効となります。変更できるのはON/OFFの論理だけです。
dx=500でない。
　　0～3の数字キィーを押すと変更できます。コマンド"SET"でこの値を変更できます。

簡単なパルス発生プログラム

必要なIOを接続してパルス発生をします。次に必要なコマンドを使用した例を示します。
それぞれのコマンドの詳細はコマンドリファレンスを参照してください。次はX軸に対してです。Y,Z,Uの各軸に対しては、Y_A,Z_A,U_Aを指定します。

```
10 PG &H410
20 ACCEL X_A -25 4000 400 100      /* 軸指定 S字の割合(25%)
                                   /* 4000PPSMAX 400(加速距離) 100(自起動)
25 RANGE X_A 50000 -50000          /*動作範囲設定 ソフトリミット
30 INSET_314 X_A SLMT_ON|LMT_ON    /*MPG-314動作フラグ設定
                                   /* "SLMT_ON" RANGEで設定した値を有効。
                                   /* "LMT_ON" リミット入力をONで有効に設定。
40 STPS X_A 0                      /*現在位置変更
50 RMVS X_A -10000                 /*相対移動(ｲﾝｸﾘﾒﾝﾀﾙ)でCCWで10000パルス発生。
60 WAIT RR(X_A)==0                 /*パルス発生の終了待ちです。
```

INSET_314 コマンドの役割は、ビット定数という MPC-684 内部で定義された値を MPG-314 の内部レジスタ WR2に設定するもので、リミット論理、インポス論理、ソフトリミットの有効、アラーム入力の論理を決定します。一度設定すると、次のINSET_314コマンドまで保持されます。
このコマンドは'上書きコマンド'です。このため、以前の設定値はクリアされます。

※注意 INSET_314について

リミット入力を無効にすることはできません。ON/OFFを選ぶだけです。

```
LMT_ON      /* リミット入力ONで検知
LMT_OFF     /* リミット入力OFFで検知
```

ソフトリミットは有効・無効の決定のみです。

```
SLMT_ON     /* ソフトリミット有効
```

【補足】入力設定時にINSET_314 ALL_A SLMT_ON|LMT_ON|MD_2PLSとすればソフトリミットは有効
INSET_314 ALL_A LMT_ON|MD_2PLSのように設定しなければ(または、設定し直せば)無効になります。

インポズ設定,アラーム設定はON/OFFを選択すると自動的に有効になります。

```
INP_ON      /* インポズ入力ONで検知
INP_OFF     /* インポズ入力OFFで検知
ALM_ON      /* アラーム入力ONで検知
ALM_OFF     /* アラーム入力OFFで検知
```

無効にするには値を設定しません。

```
INSET_314 X_A ALM_ON|INP_ON|LMT_OFF     /*ｱﾗｰﾑ,ｲﾝﾎﾟｽﾞ有効
INSET_314 X_A LMT_OFF                   /*ｱﾗｰﾑ,ｲﾝﾎﾟｽﾞ無効
```

マルチタスクでパルス発生

タスクごとに独立した軸制御することができます。次に例としてタスク1でX軸、タスク2でY軸を制御します。この例のようにMPG-314は軸ごとに異なるタスクからの制御も可能です。このため、MPG-314では搭載された4軸すべてを無駄なく使い切ることができます。

```
10 FORK 1 *YAXIS
20 PG &H410
40 ACCEL X_A -25 4000 400 100/* ↓ X軸の制御
50 RANGE X_A 50000 -50000
60 INSET_314 X_A SLMT_ON|LMT_ON
70 STPS X_A 0
80 DO
90 RMVS X_A 1000
100 WAIT RR(X_A)==0
110 RMVS X_A -1000
120 WAIT RR(X_A)==0
130 LOOP
140 *YAXIS
150 PG &H410
160 ACCEL Y_A -25 8000 400 100/* ↓ Y軸の制御
170 RANGE Y_A 50000 -50000
180 INSET_314 Y_A SLMT_ON|LMT_ON
190 STPS Y_A 0
200 DO
210 RMVS Y_A 1000
220 WAIT RR(Y_A)==0
230 RMVS Y_A -1000
240 WAIT RR(Y_A)==0
250 LOOP
```

INSET_314(条件設定) について

INSET_314はリミット、アラーム、ソフトリミット、インポジション入力の動作を設定します。このうちリミットについては、無効の設定がせできないので注意してください。設定できるのはON/OFFの論理だけです。
INSET_314コマンドはMPG-314内部のMCX-314へのレジスタ(16bit)設定コマンドです。このため指定するパラメータはビット変数となっています。複数設定する場合は、"|"演算子で結合します。また、コマンドがレジスタ設定であるために常に上書きです。

```
10 INSET_314 X_A INP_ON|SLMT_ON
20 INSET_314 X_A ALM_ON
```

前記のように実行すると最終的には20のALM_ONのみが有効となりINP_ON,SLMT_ONも無効になります。

```
10 INSET_314 X_A INP_ON|SLMT_ON| ALM_ON
```

前記のようにすればインポズ、ソフトリミット、アラームとも有効になります。

ビット定数	役割	パラメータを設定しない場合	関係するビット定数
[インポジション] INP_ON INP_OFF	サーボのインポジション入力を有効にて論理設定します	インポジション無効	
[アラーム] ALM_ON ALM_OFF	サーボからのアラーム力を有効にて論理設定します	アラーム無効	ALM@ ALM_
[ソフトリミット] SLMT_ON	ソフトリミットを有効にします。無効の場合は引数を設定しません	ソフトリミット無効	SLMP@ SLMM@ (注: Pは+,Mは-)
[(ハード)リミット] LMT_ON LMT_OFF	ハードリミットの論理を設定します。デフォルトではONでリミットとなりパルス発生しなくなります。	ONでリミット有効	LMTP@ LMTM@ LMTP_ LMTM_
[パルス出力] MD_2PLS MD_DPLS	デフォルトではMD_2PLS方式、つまりCW/CCW方式です。方向指示パルス入力の時はMD_DPLSを引数に追加します。	CW/CCW方式	

動作範囲設定(ソフトリミット)

ソフトリミットの有効無効をINSET_314の再実行で設定するのはかえって面倒です。SLMT_ONはあらかじめ設定しておいてRANGEコマンドでその都度範囲を設定して使用するほうが扱いが簡単になります。

```
10 RANGE X_A +800000,-800000          /* 実働より広い範囲
20 INSET_314 X_A INP_ON|SLMT_ON| ALM_ON
```

とすれば事実上、レンジは規制されないのと同じことになります。プログラム進行中に、必要に応じて、

```
100 RANGE X_A +1000,-2000
```

と設定すればその時点からリミットが有効に動作します。

アラーム、リミット入力の読み取り

プログラム動作中に各入力が検出されたかどうかは次のようにRR()関数をもちいます。たとえば、ソフトリミットが動作した状態では、RR(X_A,SLMT@)の値が0でなくなります。読み取りビット定数には末尾に'@'のついたものと'_'ものがあり意味が異なります。@のついたものは状態監視読み取り、つまり、その入力ポートが現在どのような状態になっているかを返すものです。ALM_ONと設定されていて入力ポートをONにするとRR(X_A,ALM@)は0でない値を返します。これに対して、ALM_等の_のついたビット定数は、パルス発生が停止した原因を知らせるためのものです。たとえば、ALM_ON を設定し、入力を実際に ON すると RR(X_A,ALM@)は0でない値を返しますが、RR(X_A,ALM_)は実際にパルス発生をして停止させるまでは0のままです。読み取りは、ビット定数をORすることもできます。この場合は、いずれかの入力を検出すると0でなくなります。

```
RR(X_A,ALM@|SLMP@|SLMM@)
```

しかし、ALM@|LMTP_|LMTM_というような種類の異なるビット定数を混在させることはできません。

INCHK_314 について

配線直後に入力ポートの様子を確認するコマンドです。実行すると次のように表示されます。
X_S1～U-INPまでは、実際の入力論理値を表示しますが、XLMT+～ULMT-では、設定した論理値にしたがってリミットが動作している場合にONと表示します。
パワーオン後、何も設定しないでこのコマンドで LMT を確認すると、ON で有効というデフォルト状態になっているために、LMTポートをONするとONと表示します。

```
#INCHK_314
X_S1 __ Y_S1 __
Z_S1 __ U_S1 __
```

```
X_S2 __ Y_S2 __
Z_S2 __ U_S2 __
XIN2 __ YIN2 __
ZIN2 __ UIN2 __
XIN3 __ YIN3 __
ZIN3 __ UIN3 __
X-INP __ Y-INP __
Z-INP __ U-INP __
X-ALM __ Y-ALM __
Z-ALM __ U-ALM __
XLMT+ __ XLMT- __
YLMT+ __ YLMT- __
ZLMT+ __ ZLMT- __
ULMT+ __ ULMT- __
```

多軸同時制御と直線補間

■RMVS,MOVSコマンド

4つのパラメータを同時に指定してパルス発生することができます。動作させない軸には0をセットします。

```
RMVS 1000 0 2000 0          /*XU軸のﾊﾟﾙｽ発生をします。
MOVS 1000 2000 VOID VOID    /*絶対位置管理のﾊﾟﾙｽ発生の場合は動作させない軸に
                            /*対しては、VOIDを設定する。
                            /*(0では0位置を意味するため)
```

■RMVL,MOVLコマンド

直線補間の場合は、RMVL,MOVLを用います。この場合はMPG-314の機能上三軸までとなります。四軸直線補間はできません。

```
RMVL 1000 2000 3000 0       /* XYU直線補間　X軸の速度に従います。
MOVL VOID 1000 2000 3000    /* YUZの直線補間　Y軸の速度に従います。
```

■原点復帰

原点復帰にはJ4のIN0,IN1を使用します。IN0をニアオリジンとしIN1をサーボのZ(C)相に接続します。
IN1側はJ2(23～26)に設けられた、別端子を使用することにより差動入力対応とすることができるためです。
HOMEには次のようにHOMEコマンドを使用します。条件設定はビット設定を用います(IN0_ON/OFF～IN3_ON/OFF)。
条件はOR条件です。またHOMEで設定された停止条件は残ってしまいますので、STOPコマンドでクリアする必要があります。

```
HOME X_A -1000 IN0_ON|IN1_OFF       /* IN0がONもしくはIN1_OFFになったら停止
```

通常、ニアオリジン、オリジンという順になりますので次のようになります。

```
HOME X_A -1000 IN0_ON     /* スリット検出
WAIT RR(X_A)==0
HOME X_A -100 IN1_ON      /* Z相検出
WAIT RR(X_A)==0
STOP X_A VOID             /*停止条件クリア
STPS X_A 0                /*現在位置クリア
```

また、HOMEコマンドは二軸まで対応しますので次のようにも記述できます。

```
10 HOME X_A -1000 IN0_ON Y_A -1000 IN0_ON    /* ｽﾘｯﾄ検出
20 WAIT RR(X_A|Y_A)==0
30 HOME X_A -100 IN1_ON Y_A -100 IN1_ON      /* Z相検出
40 WAIT RR(X_A|Y_A)==0
50 STOP X_A|Y_A VOID                         /*停止条件クリア
60 STPS X_A 0 : STPS Y_A 0                   /*現在位置クリア
```

ニアオリジンまでの高速原点復帰は次のような記述ができます。

```
10 STOP X_A IN0_ON          /*停止条件＝スリット検出
20 RMVS X_A 10000           /*10000パルス発生までに上記条件が成立したら減速停止
30 WAIT RR(X_A)==0          /*停止待ち
40 STOP X_A VOID            /*条件クリア
```

溜まりパルスのリセット

原点復帰速度が十分に低速で、機械系のバックラッシュなどが少なければ前記のような原点復帰で十分なのです。しかし、溜まりパルス(サーボ追従の遅れ)がある場合、原点復帰の成立と同時に溜まりパルスリセット信号を出力する必要があります。

```
10 '原点復帰で残パルスクリアの例
20 'Near Orgin
30 HOUT &HF0 : ' Servo On Servo reset OFF
40 HOME X_A 100 IN0_ON
50 HOME Y_A 100 IN0_ON
60 HOME Z_A 100 IN0_ON
70 HOME U_A 100 IN0_ON
80 WAIT RR(ALL_A)==0
90 'Detect Z phase
100 '必要に応じて他タスクpause
110 ' pause 3 4 5
120 HOME X_A 10 IN1_ON
130 HOME Y_A 10 IN1_ON
140 HOME Z_A 10 IN1_ON
150 HOME U_A 10 IN1_ON
160 a=0
170 'RR(ALL_A)の値をHOUTに反映
180 'Zのビットが4Uが8となるので注意
190 '論理反転してリセット信号とする。
200 WHILE a<>&HFF : HOUT a=(15-RR(ALL_A))|&HF0 : WEND
210 TIME 10
220 HOUT &HF0 : 'カウンタリセットクリア
230 ' cont 3 4 5
```

ここではOP1～OP4をサーボリセット信号(X,Y,Z,Uの順)として使用します。MPG-314のビジービットの値を反転して、OP1～OP4に複写するという方法です。速度を早めるために一行の式で記述しています(ステップ200)。また、動作している他のタスクは応答速度を悪化させますので、出来る限りPAUSEなどで停止させて下さい。

カウンタ機能

MPG-314には4軸のロータリーエンコーダ用カウンタ機能が付属しており、OC5～OC8にフォトカプラを装着することにより使用可能となります。
カウンタに関係するコマンドと関数は次のとおりです。

◆STPS AXIS Val

AXIS：カウンタ指定 X_C～U_C
Val ：設定したい値。
注意: AXISにX_A～U_Aを用いると現在位置の意味になります。

◆X(-1)～U(-1)

座標関数に-1を設定するとカウンタ値を返します。

◆X(-2)～U(-2)

座標関数に-2を設定するとカウンタ値を返し、現在のカウンタをクリアします。

X(-2,Fac)～U(-2,Fac) のように引数を追加すると"カウンタ値×Fac"を返します。-1の場合も有効です。
次のプログラムは、エンコーダをJOGダイヤルに使用した例です。Aに倍率を入れておきます。Aの値を多くすれば移動量が多くなります。このように関数とコマンドを一行にまとめてしまうと高速になります。
480のスワップは、カウンタが変化しない場合、無限ループ実行になり全体の実行効率が低下するのを回避するためです。なおカウンタの値をまとめてクリアする場合は、

```
STPS X_C|Y_C 0              /* XY同時クリア
STPS ALL_C 0                /* 全クリア
```

という記述ができます。

```
--初期化--
10 PG &H410
20 PG &H410 1
30 A=1
40 INSET_314 X_A|Y_A 0
----------
330 ACCEL 8000
340 STPS X_C 0
350 STPS Y_C 0
360 FORK 1 *jog2
370 END

440 *jog2
450 WHILE A<>0
460 RMVS X(-2,A) Y(-2,A) 0 0
470 WAIT RR(X_A|Y_A)==0
480 SWAP
490 WEND
500 PRINT "Jog End"
#
```

注(1):
10～40は初期値として必要な設定です。INSET_314はカウンタをA-B2相方式に設定します。この例ではダミーとして0を引数としていますが、パラメータはどのような値でもかまいません。INSET_314が実行されていないと2相信号をカウントしません。

注(2):
カウンタはデフォルトで"4てい倍"となっています。(2倍,1倍)率を与えるにはINSET_314の設定値に&H400(2倍)、&H800(1倍)をORします。

```
INSET_314 X_A &H800|INP_OFF          /*1倍で読み取り
```

条件停止

STOPコマンドには二通りの使用方法があります。
ひとつはインタプリタ上で、適宜 STOP コマンドを実行しプログラム制御でパルス発生を停止する場合です。もうひとつは、IN0～IN3のハードで用意された自動停止を使用する場合です。

■ソフト停止

◆STOP AXIS CMND

AXIS : X_A ～ U_Aを指定します。
CMND : STP_D(減速停止),STP_I(急停止)のいずれかを設定します。

```
10 PG &H410
20 ACCEL X_A 4000
30 CLRPOS
40 MOVS X_A 10000
45 WHILE RR(X_A)<>0
50 IF SW(1)==1 THEN : STOP X_A STP_D : END_IF
60 WEND
```

前記の例は動作中にセンサ入力を検出したら減速停止させるというものです。

■ハード入力検出停止

◆STOP AXIS cond

インタプリタで制御する停止は自由度が高いのですが、応答速度に限界があり、正確な検出停止には使えません。MPG-314のJ2,J4に用意されているIN1～IN3の入力を用いるとハード的に入力を検出して停止するため、正確さ高速さを要求される停止機能に対応します。

```
40 ACCEL X_A 4000
50 CLRPOS                /* 現在値クリア
60 STOP X_A IN3_ON       /* STOPの条件設定
70 MOVS X_A 10000        /* パルス発生
80 WAIT RR(X_A)==0       /* 停止待ち
90 STOP X_A VOID         /* STOPの設定を解除
```

```
100 PRINT X(0)              /* 現在値表示
110 END
```

この例はJ2側のIN3がONになったらパルス発生を停止します。停止したら必ずSTOPの設定を解除します。
条件はOR演算子"|"で結合できますが、AND条件はICの機能に無いためサポートされていません。

予約定数

■指定ｸﾞﾙｰﾌﾟ

NOP	ﾉｰｵﾍﾟﾚｰｼｮﾝ。軸選択のﾚｼﾞｽﾀ設定に使用。
VOID	入力無効定数
CLR_ERR	ｺﾏﾝﾄﾞｴﾗｰ解除
STP_I STP_D	停止ｺﾏﾝﾄﾞ STP_I：急停止　STP_D：減速停止
X_A,Y_A, U_A,Z_A, ALL_A	軸指定定数 X_A：X軸　Y_A：Y軸　U_A：U軸 Z_A：Z軸　ALL_A：全軸 (ACCEL等で)
X_C,Y_C, U_C,Z_C	カウンタ指定定数 X_C：X軸　Y_C：Y軸 U_C：U軸　Z_C：Z軸
CW CCW	円弧補間方向指定。連続補間に使用。 CW：時計方向　CW：反時計方向
DS_DACL EN_DACL	自動減速無効、有効。連続補間に使用。 DS_DACL：無効　EN_DACL：有効
IN0_ON,IN1_ON,IN2_ON,IN3_ON, IN0_OFF,IN1_OFF,IN2_OFF,IN3_OFF	停止入力をON/OFF状態で有効にする。 HOME X_A,STOP X_Aﾀｲﾌﾟのｺﾏﾝﾄﾞで使用。
INP_ON INP_OFF	ｲﾝﾎﾟｼﾞｼｮﾝを有効にしてON/OFFを決定する。 INP_ON：lowでｱｸﾃｨﾌﾞ　INP_OFF：Hiでｱｸﾃｨﾌﾞ
ALM_ON ALM_OFF	ｱﾗｰﾑを有効にしてON/OFFを指定 ALM_ON：lowでｱｸﾃｨﾌﾞ　ALM_OFF：Hiでｱｸﾃｨﾌﾞ
LMT_ON,LMT_OFF	ﾘﾐｯﾄONの時に成立
SLMT_ON	RANGEｺﾏﾝﾄﾞによるｿﾌﾄﾘﾐｯﾄの有効化定数
MD_2PLS	CW/CCWﾊﾟﾙｽ発生指定
MD_DPLS	DIRﾊﾟﾙｽ発生指定

■状態監視ｸﾞﾙｰﾌﾟ

EMG@	EMGがｲﾈｰﾌﾞﾙ検出
ALM@	ｻｰﾎﾞｱﾗｰﾑ出力ON検出
LMTP@	ﾘﾐｯﾄ入力(+)検出
LMTM@	ﾘﾐｯﾄ入力(-)検出
SLMP@	ｿﾌﾄﾘﾐｯﾄ(+)検出
SLMM@	ｿﾌﾄﾘﾐｯﾄ(-)検出

■停止条件ｸﾞﾙｰﾌﾟ

EMG_	EMGによって停止
ALM_	ｱﾗｰﾑ入力によって停止
LMTP_	ﾘﾐｯﾄ入力(+)によって停止
LMTM_	ﾘﾐｯﾄ入力(-)によって停止
IN3_	STOP条件のIN3によって停止
IN2_	STOP条件のIN2によって停止
IN1_	STOP条件のIN1によって停止
IN0_	STOP条件のIN0によって停止

レジスタ概要

■読み取り用

◆RR0:パルス、エラー監視
パルス発生中の監視、エラーの監視用です。関数RR(0)で参照することができます。下位4ビットがビジー、次の4ビットがエラーです。次のように使用します。エラーは次のRR1の上位8ビットの反映です。
パルス発生の終了待ち

```
WAIT RR(0)&X_A=0
```

もしくは

```
WAIT RR(X_A)=0
```

他のタスクのRR0の読み取り

```
RR(&H410)
```

◆RR1:軸毎のエラーステータス1
X,Y,U,Zそれぞれか持つステータスです。関数RR(軸定数,1)という形式で読み取ることができます。このレジスタの上位8ビットはパルス発生がどのように終了したかを表します。RR()関数で読み取りますが次のような使用方法となります。
X軸リミットエラーの参照

```
LMT=&h3000&RR(X_A,1)
```

◆RR2:軸毎のエラーステータス2
X,Y,U,Zそれぞれが持つエラー入力の参照レジスタです。ALMやLMT信号を直接参照するにはこのレジスタを参照します。

◆RR3:軸毎のエラーステータス3
X,Y,U,Zそれぞれが持つカウンタの比較情報です。MCX-314には軸ごとに比較用レジスタと比較器があります。これにより特定の場所で正確な信号を発生させることも可能です。

■設定用

◆WR0:コマンドレジスタ
軸選択とMCX-314に対するコマンドレジスタです。例えばHOUT X_A;STP_Dという記述では、X_A;STP_Dという演算によって&H0126という数値が生成されますがHOUTはこれをWR0への直接書き込みとして扱います。WR0は軸選択レジスタとしても使用されるため、次のWR1,WR2への書き込み時にもダミーライトが必要となります。

◆WR1:モードレジスタ1
主に IN0 ～ IN3 のドライブ停止検出信号の設定として使用します。WR1 への書き込みは次のようなフォーマットになります。

```
HOUT 2 (X_A;NOP,1)
```

2はWR2に書き込む値でIN0がローで停止という条件です。()のの中の二番目の引き数X_A;NOPでWR0に対して軸設定をします。','の後ろの番号がレジスタ番号です。レジスタ番号にボードアドレスを加えると任意のアドレスのMPG-314にアクセスすることができます。

◆WR2:モードレジスタ2
リミット検出やインポジション検出の設定を行うレジスタです。しかしながらこのレジスタにはパルス出力モードの設定ビットも含まれていますので注意して再設定してください。

```
HOUT INP_ON+MD_2PLS (X_A;NOP,2)
```

ここではINP_ON+MD_2PLSをWR2に設定する値としています。MD_2PLSはパルス出力モードをCW/CCW負論理方式(デフォルト)に設定するビット定数ですが、もしこの値を加えておかないとパルス出力のモード設定フィルードがすべて0となり、パルス出力が正論理に変わってしまいます。(X_A;NOP,2)はWR1の場合と同様の入力でX軸選択でWR2を選んでいます。

◆WR3:モードレジスタ3

下位3ビットが加減速度のモード設定となります。初期値としてこのレジスタは4(自動減速、対称減速、S字加減速度有効)が設定されています。このほかにはOUT4～7の出力モード設定があり汎用出力として使用するか、位置信号の出力として使うかを決定できます。

WR3のビット配置

0000 ¦ 7 6 5 4 ¦ 0 X X E1 ¦ E0 S D1 D0 ¦

上位4ﾋﾞｯﾄ:無効

7～4：出力ﾎﾟｰﾄ(J2の17～22)。下位2ﾋﾞｯﾄのみ使用

E1,E0:MPG-314では非接続なので0とする。

S:S字有効=1

D1:減速レジスタの選択。通常0

D0:自動加減速の有効・無効。1で無効。

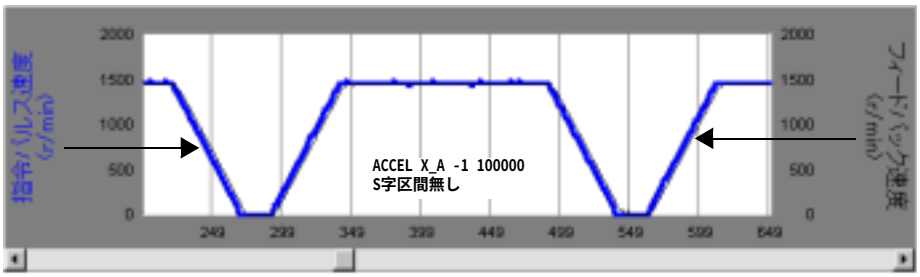
S字加減速

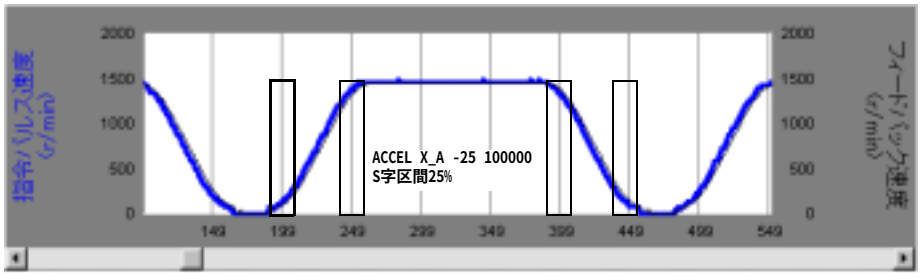
S字加減速を使うと滑らかな始動・停止が出来て、速くインポジションを得られる場合があります。しかし設定を大きくすると加減速領域が広がり、かえってタクトタイムが長くなることもあります。適切にドライバのゲイン調整を行い装置剛性に合ったS字領域を設定しなければなりません。次はMPG-314でS字加減速をした時の実測例です。

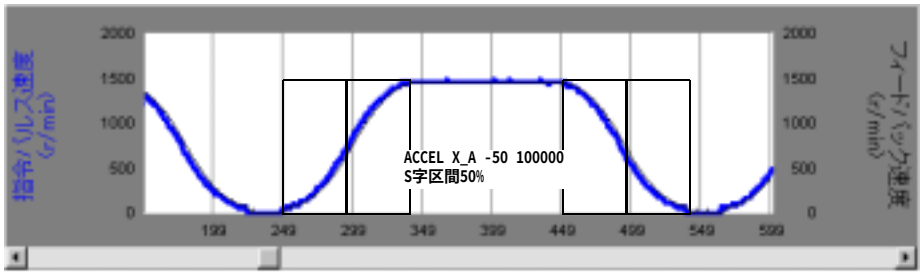
ｻｰﾎﾞﾓｰﾀ	安川 SGM-A3BW12C
ｻｰﾎﾞﾊﾟｯｸ	安川 SGDA-A3BP
	ﾓｰﾀは無負荷
ﾓﾆﾀﾘﾝｸﾞｿﾌﾄ	Sigma Win Ver1.20

ﾌﾟﾛｸﾞﾗﾑ

```
10  ACCEL X_A -1 100000   /*第2ﾊﾟﾗﾒｰﾀを変える
20  DO
30    RMVS X_A 40000
40    WAIT RR(X_A)==0
50    TIME 50
60  LOOP
```

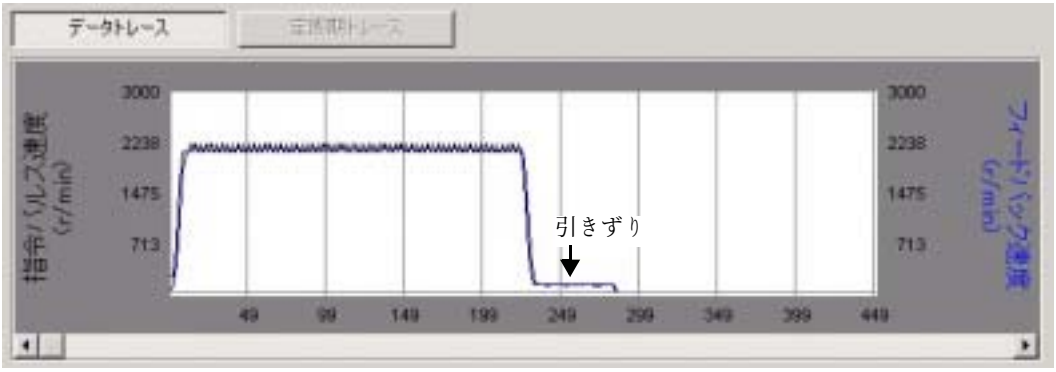






引きずり現象について

引きずり現象とは、次の図のように減速が早めに始まり、最低速度でのパルス発生時間が異常に長くなることをさします。この現象は、S字加減速を設定した時のみに発生するもので、FEED値を大きくとり最低速度と最高速度の差が少なくなったとき発生しやすい、という傾向があります。台形駆動では、この現象はありません。次の例は、ACCEL X_A -10 250000 20000 2000 : FEED X_A 220 での実測で、0.5秒程度の引きずりを発生しています。



■原因

MPG-314に使用しているIC(MCX-314)の内部仕様に起因しており、メーカでは将来的には対処修正することとしていますが、対応時期については明確にしていません。

■検出方法

引きずりの発生は、数学的な方法で系統的に見つけることはできないというのがメーカの見解です。このため、MPG-314のパルス発生を実際に計測しました。パルス発生中の314のR1レジスタを監視することによって減速後の定速度移動時間を計り、それを引きずり時間としました。次がそのプログラムです。
このプログラムでは、0.2秒以上、最低速度移動をした場合を引きずりとしています。また、プログラム冒頭の”WOW &H8000 &H400 “ は、314のパルス発生ポートを無効にし装置が動作しないようするためのものです。

```
  WOW &H8000 &H400 : PG &H400 : PG &H400 1
    Smode=10 :   Max=250000 :   Leng=20000 :   Min=2000 :   Feed=210
'
  ACCEL X_A -1*Smode Max Leng Min
  FEED X_A Feed
  CLRPOS
  RMVS X_A Leng*2
  FORK 1 *moni
  WAIT RR(X_A)==0
  QUIT 1
  IF kk<20 THEN
    PRINT "OK  FEED " Feed
    ELSE  : PRINT "Hikizuri =" kk*10 "msec @FEED" Feed
  END_IF
  END
*moni
    kk=0
  WAIT RR(X_A,1)&16<>0 : PRINT "catch" : '減速検出
  WAIT RR(X_A,1)&16==0 :        '減速終了
*watch
  IF RR(X_A,1)&12==8 THEN :   kk=kk+1 : END_IF
  TIME 10
  IF RR(X_A)<>0 THEN : GOTO *watch : END_IF
  END
```

■結果

様々なパタンで計測した結果、S字引きずりが発生しやすいのは、FEED値を200前後まで大きくした場合であることがデータから推察されます。また、S字を50%とすると例外的に、FEED 96で0.2秒の引きずりが発生

しますが、これを引きずりと考えるか、大きなS字の一部と考えるかは微妙な問題です。また、ACCEL設定のみしてFEEDを用いない場合は、引きずりの発生は少なくなります。

■対応

前記のことより、S字を用いる場合、FEED設定による遅い速度は使用しないで、ACCELの再設定を実施して下さい。(S字が20%以下で100kppsから500kppsの間”FEED 0”では0.1秒以上の引きずりは観測されません。またMPG-314のACCEL設定時間はわずかに500μ秒です。)

■計測データ

次は同一条件で、移動距離を変えて引きずりを計測しました。

ACCEL -10 25000 20000 2000 P=60000	ACCEL -10 25000 20000 2000 P=40000	ACCEL -10 25000 20000 2000 P=42000
FEED/ msec	FEED/ msec	FEED/ msec
210 580 211 340 214 500 220 300 228 240 234 210	210 580 211 340 214 500 220 310 228 240 234 220	210 590 211 330 214 500 220 300 228 240 234 210

結果:引きずりは距離に関係なく同程度発生します。(ただし、短い距離では発生しません)

次は最低速度を変えました。(最低速度を上げれば当然引きずり時間は減ります)

ACCEL -10 25000 20000 3000 P=60000	ACCEL -10 25000 20000 3000 P=40000	ACCEL -10 25000 20000 3000 P=42000
FEED/ msec	FEED/ msec	FEED/ msec
212 280 215 260 225 220	212 280 215 260 225 230	212 290 215 250 225 230

前記の例により、距離の変動には関連が無いと推察されます。

次は50kppsから500kpps(10kppsきざみ)でのFEED 0での引きずり計測例です。ここでは0.1秒以上の引きずりを計測しました。太字が問題となる引きずりです。Sは5~50まで変化。

ACCEL -S Max 30000 1000	ACCEL -S Max Max/10 1000
S% Max[pps] 引きずり時間(mSEC) 加速距離[pls]	S% Max[pps] 引きずり時間(mSEC) 加速距離[pls]
速度に対して加速度が小さい(加速距離が長い)場合に引きずりが発生しやすいです。	速度と加速が適切な値であれば、FEED 0ではほとんど引きずりは発生しません。
30 80000 430 30000 **30 90000 270 30000** **30 100000 240 30000** **30 110000 230 30000** **30 120000 250 30000** **30 130000 210 30000** **30 140000 200 30000** 30 150000 100 30000 30 170000 100 30000 30 180000 110 30000 30 220000 170 30000 30 230000 110 30000 30 300000 100 30000 30 330000 100 30000 30 350000 100 30000 30 410000 120 30000 35 340000 130 30000 35 370000 120 30000	25 360000 100 36000 25 420000 110 42000 25 430000 100 43000 25 490000 150 49000 25 500000 110 50000 30 300000 100 30000 30 340000 120 34000 30 350000 110 35000 30 390000 140 39000 30 400000 120 40000 **30 410000 100 41000** 30 460000 210 46000 30 470000 160 47000 30 480000 150 48000 30 490000 110 49000 30 500000 170 50000 35 380000 110 38000 35 460000 130 46000

応用例

■一枚のMPG-314の軸をタスクごとに非同期で使用する場合

MPG-68K/405では1枚のMPGに対して複数のタスクから同時にパルスコマンドを実行できませんがMPG-314では可能です。次のプログラムは1枚のMPGの各軸を別々のタスクで制御しています。
MPG-68K/405でこのような制御を行おうとすると軸数分のMPGが必要だったので省スペース、低コストになります。

```
  PG &H410 1            /*複数のタスクを一枚のMPGに割り当てます。
  PG &H410 2
  PG &H410 3
  PG &H410 4
  FORK 1 *MPG_x         /*それぞれのタスクを起動します。
  FORK 2 *MPG_y
  FORK 3 *MPG_u
  FORK 4 *MPG_z
  END
*MPG_x
  ACCEL X_A 8000        /*加速度を軸毎に選択します。
  STPS X_A 0
  DO
    GOSUB *PULSE X_A    /*タスク間共通サブルーチンをコールしていますが、
  LOOP                  /*例文を簡略化するためです。
*MPG_y
  ACCEL Y_A 6000
  STPS Y_A 0
  DO
    GOSUB *PULSE Y_A
  LOOP
*MPG_u
  ACCEL U_A 4000
  STPS U_A 0
  DO
    GOSUB *PULSE U_A
  LOOP
*MPG_z
  ACCEL Z_A 2000
  STPS Z_A 0
  DO
    GOSUB *PULSE Z_A
  LOOP
*PULSE                  /*タスク間で共通のルーチンです。
  _VAR axs!             /*そのためにローカル変数を用いています。
  FOR i!=0 TO 10        /*単軸制御は例のように軸選択定数を用いて制御するのが安全です。
    FEED axs! i!*20
    RMVS axs! 1000
  NEXT i!
  WAIT RR(axs!)==0
  MOVS axs! 0
  RETURN
```

■一枚のMPG-314の軸で三軸直線補間と残りの軸を他のタスクで制御

次のように直線補間と単軸駆動を組み合わせて非同期で使用できますが､XY,ZUように二軸直線補間を二組同時に使用することはできません。

```
  PG &H410 1            /*二つのタスクを一つのMPGに割り当てます
  PG &H410 2
  FORK 1 *MPG_xyz
  FORK 4 *MPG_z
  END
*MPG_xyz
  ACCEL X_A 8000        /*主軸(X>Y>Z>U)のACCELを設定
  STPS 0 0 0 VOID       /*他のタスクが使う軸はVOID変数で扱わないようにする。
```

```
  DO
    FOR i=1 TO 10
      RMVL i*1000 i*100 i*-10 0        /* RMVL,MOVLは直線補間
    NEXT I
    WAIT RR(X_A)==0
    MOVL 0 0 0 VOID
  LOOP
*MPG_z                    /* Z軸の単独制御
  ACCEL Z_A 2000
  STPS Z_A 0
  DO
    GOSUB *PULSE Z_A
  LOOP
*PULSE
  _VAR axs!
  FOR i!=0 TO 10
    FEED axs! i!*20
    RMVS axs! 1000
  NEXT i!
  WAIT RR(axs!)==0
  FEED axs! 0
  MOVS axs! 0
  RETURN
```

■パルス発生　途中停止や速度変更

入力を検出してソフトで停止させる

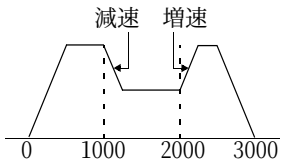
MPG-314 ネイティブコマンドは実行後すぐに次のステップに移行します。次はパルス発生しながらセンサがオンするのを待ちます。このようにパルス発生とその後の管理が同一タスクで行えるのでプログラムの見通しがよくなります。

```
*Y
RMVS Y_A 200000           /*相対座標移動
WAIT HPT(1)=1             /*センサ入力待ち
STOP Y_A STP_I            /*STP_Iで即停止、STP_Dで減速停止です。
WAIT HPT(1)=0
MOVS Y_A 0
WAIT RR(0)&Y_A=0
TIME 500
GOTO *Y
```

条件により速度を変更する

行き先の決まっているパルス発生でS字加減速を設定していると減速のみで増速できません。RMVCではS字加減速でも増速・減速いずれも可です。

```
PG &H410
ACCEL 8000
CLRPOS
RMVS X_A 3000
WAIT X(0)>1000
FEED X_A 128              /*速度を途中で変更する。
WAIT X(0)>2000
FEED X_A 0
```



原点復帰

原点復帰はHOMEコマンドを使用すると明示的になりますが、現在位置については放置されます。原点復帰が完了したら位置をクリア、あるいは設定する必要があります。

```
PG &H410
ACCEL 8000
HOME X_A IN0_ON -125 Z_A IN0_ON -200   /*速度指定原点復帰 IN0使用 FEED 125 CCW FEED
WAIT RR(X_A|Z_A)==0                      200 CCW
HOME X_A -1000 IN1_ON Z_A -500 IN1_ON  /*定則原点復帰 IN1使用
WAIT RR(X_A|Z_A)==0
PRSET_ACCEL X_A Z_A                    /* ACCELパラメータ復帰 停止条件クリア
```

```
FEED X_A 0 Z_A 0                    /* HOMEで変更したFEEDの変更
STPS 1000 VOID VOID 1000            /*位置設定
```

■パルス発生　途中停止　ハード

MCX-314には入力ポートIN0-IN3を使用した停止条件を決定するレジスタWR1があります。このレジスタに適切な値を設定することにより、各軸ごとに停止条件を定めることができます。IN0,IN1はS1,S2に割り当てられ、IN2は差動入力ポート、IN3は拡張入力ポートに割り当てられています。これにより必要に応じてハード停止させることができるためホスト側の遅れ時間を気にすることなく確実にパルス発生を停止させることができます。このWR1を設定するためにSTOP拡張コマンドとIN0_ON～IN3～OFF定数が用意されています。

```
PG &H410
ACCEL 10000
STPS X_A 0
STOP X_A IN3_ON|IN1_OFF  /* IN1がOFFになるかIN3がオンになると停止
RMVS X_A 1000000
STOP X_A VOID            /*停止条件の解除
```

INPOSの有効 サーボオンエラーリセットなど

INPOSを有効にするとｻｰﾎﾞのINPOS信号と連動させることができます。INPOSの設定はWR2レジスタで行います。レジスタの意味は次のとおりです。なおこのレジスタを操作するのに必要な定数セットが用意されています。

MD_2PLS MD_DPLS	デフォルトのCW/CCW負論理パルス方式です。 DIR/PULSE方式です。
INP_ON INP_OFF	インポジションをON入力待ちとします。 インポジションをOFF入力待ちとします。
ALM_ON ALM_OFF	アラームをONで発生とします。 アラームをOFFで発生とします。
LMT_ON LMT_OFF	リミット信号をONで発生とします。 リミット信号をOFFで発生とします。

X軸のインポジションをONで有効、ALMをONで有効、LMTをONで検出とするには

```
wr2=MD_2PLS|INP_ON|ALM_ON|LMT_ON
hout wr2 (X_A;NOP,2)
```

wr2を設定する場合はMD_2PLSをかならず加えて設定しなおしてください。また、パルスDIR/PULSE方式にするにはMD_2PLSの替わりにMD_DPLSを使います。

Z軸のINPOSを有効とするには

```
10 PG &H410
20 ACCEL 10000
30 CLRPOS
35 HOUT MD_2PLS|INP_ON  (Z_A;NOP,2)
40 DO
50 RMVS Z_A 10000
60 RMVS Z_A -10000
70 LOOP
```

■移動距離を規定しないパルス発生（RMVC,PLSC)

原点復帰やテープの定速送りでパルス量を規定しないパルス発生があります。この場合はRMVC,PLSCを用います。RMVCはACCELの指定に従います。減速停止と組み合わせて次のように使用できます。また原点復帰の場合座標値のクリアが必要ですが、STPS X_A 0、STPS VOID 0 0 VOIDなどのように実施します。

```
PG &H410
ACCEL 10000
FEED X_A 64 : FEED Z_A 32
RMVC Z_A 1 X_A -1        /* ZをCW,XをCCWで回転。ACCELに従って加速、FEEDに従った速度
TIME 1000
```

```
STOP Z_A STP_D              /* Zを減速停止
TIME 1000
STOP X_A STP_I              /* Xを急停止
```

次はIN0～IN3を用いた例です。

```
PG &H410
ACCEL 10000
FEED X_A 64 : FEED Z_A 32
STOP X_A IN0_ON             /* X軸停止設定XS1
STOP Z_A IN0_ON             /* Z軸停止設定ZS1
RMVC Z_A 1 X_A -1           /* XS1(IN0)検出減速停止
WAIT RR(X_A|Z_A)==0         /*停止待ち
STOP X_A VOID               /*停止設定解除
STOP Z_A VOID               /*停止設定解除
```

RMVCの替わりに PLSCを用いると加減速なしの定速パルス発生になり、 ppsを直接指定できます。
しかし､PLSCコマンドはACCELの設定を壊してしまうため､実行後PRSET_ACCELコマンドでACCEL設定値を戻す必要があります。 PRSET_ACCELには設定された停止条件を解除する機能も含まれています。

```
PG &H410
ACCEL 10000
STOP X_A IN0_ON
STOP Z_A IN0_ON
PLSC Z_A 100 X_A -100       /*100pps指定
WAIT RR(X_A|Z_A)==0         /*停止待ち
PRSET_ACCEL X_A Z_A         /*パラメータ復帰。また停止条件解除
```

■エラーの扱い方

ERR_PAUSEを使用すると発生したエラーに対応してタスクを自動停止させることができます。PG出力でエラーとなった場合、パルス発生そのものはされませんが、インタプリタは動作しつづけます。このため、アプリケーションによっては装置を誤作動させてしまう場合があります。ERR_PAUSE をセットしておけば確実にインタプリタが停止しますので安全な対応をとることができます。

```
10 PG &H410 1
20 FORK 1 *prg
30 DO
40 IF TASK(1)==-3 THEN                  /*停止を検出
50 PRINT "Err happen!!" : TIME 1000
60 HOUT X_A;CLR_ERR &H410               /*エラーのクリア
70 CONT 1                               /*（実際にはもう少し複雑な手続きの上、
80 END_IF                               /*  再起動か一時停止の解除とする）
90 LOOP
100 END
110 *prg
120 ACCEL 8000
130 wr2=MD_2PLS|ALM_ON                  /*ALAM信号を有効にします。
140 HOUT wr2 (X_A;NOP,2)                /*X軸のWR2を書き換えます。
150 ERR_PAUSE X_A                       /*X軸のエラー検出をイネーブルします。
160 DO
170 RMVS X_A 1000                       /*実行プログラム
180 WAIT RR(X_A)==0
190 RMVS X_A -1000
200 WAIT RR(X_A)==0
210 LOOP
```

RS-232のプログラム

MPC同士での通信例

最近のシステムはますますインテリジェント化され画像処理装置、表示装置、パソコン、ＬＡＮなどの外部機器と通信で結ばれ多くのデータの授受が行われています。MPC-684にはRS-232ポートが3チャンネル装備されておりそのうち2チャンネルをユーザーが使用できます。PRINT文やINPUT文による文字列単位の送受信、INP$・PUT#によるキャラクター単位の送受信、さらにLEN・STR・STRCPY・VALなどの関数・コマンドによる柔軟な文字列処理で複雑な通信フォーマットにも対応できます。次のプログラムはMPC-684 2台のCH0同士を接続し、ＭＰＣ－Ａから送信されるデータをＭＰＣ－Ｂで解析しています。

■ＭＰＣ－Ａ（送信側）

送信フォーマット

［SX］［変数名］［*］［データ］［*］［チェックサム］［EX］［ET］

概 要

最初にEQコードを送信しAKコードの応答を待つ（受信側のREADY確認）。その後フォーマットに従い変数名・データ・チェックサムを送信する

```
  CNFG#0 "9600b8pns1XON"
'
  PRINT#0 CHR$(&H5)                 'EQコード送信
  WAIT INP$#0(1)==CHR$(&H6)         'AKコード受信待ち
    ABCDE=12356
    FGHIJ=98765
    A$="FGHIJ"                      'A$に文字列FGHIJを入れる
    B$=STR$(FGHIJ)                  'B$にFGHIJの値を文字列として格納
  PRINT#0 CHR$(&H2) A$ "*" B$ "*"   'SXコードに続きデータを出力
    CS=0
  FOR I=0 TO LEN(A$)                'A$のチェックサムを計算
    STRCPY A$ C$ I 1
     CS=CS+ASC(C$)
  NEXT I
  FOR I=0 TO LEN(B$)                'B$のチェックサムを計算
    STRCPY B$ C$ I 1
      CS=CS+ASC(C$)
  NEXT I
    CS=CS+ASC("*")
    CS$=HEX$(CS&HFF)                'チェックサムの値を文字列としてC$に格納
  PRINT#0 CS$ CHR$(&H03)            'C$とEXコードを出力
  PRINT#0 CHR$(&H4)                 'ETコードを出力
```

■ＭＰＣ－Ｂ（受信側）

受信フォーマット

［SX］［変数名］［*］［データ］［*］［チェックサム］［EX］

概　要

EQコードを受信したらAKコードを返してREADY状態を知らせ受信を開始する。EXコード受信によりチェックサムデータを比較、ETコードで終了する。

```
  CNFG#0 "9600b8pns1XON"
'
  WAIT INP$#0(1)==CHR$(&H5)         'EQコード待ち
  PRINT#0 CHR$(&H6)                 'AKコード出力
     A$=""
*LOOP
  WAIT LOF(0)<>0
    A$=INP$#0(1)
  SELECT_CASE A$                    'SXコードなら受信ETコードなら終了
        CASE CHR$(&H2) : GOTO *CASE1
```

```
        CASE CHR$(&H4) : GOTO *CASE2
  END_SELECT
  GOTO *LOOP
*CASE1
    C=0
    A$=""
    B$=""
    C$=""
  DO UNTIL A$=="*"                    ’変数名を受信
      B$=B$+A$
      A$=INP$#0(1)
      C=C+ASC(A$)                     ’チェックサムデータ加算
  LOOP
  PRINT B$
    A$=""
  DO UNTIL A$=="*"                    ’データを受信
      C$=C$+A$
      C=C+ASC(A$)
      A$=INP$#0(1)
  LOOP
  PRINT C$
    C=C&&HFF
    D$=HEX$(C)                        ’チェックサムデータを文字列に変換
    A$=INP$#0(2)                      ’チェックサムデータを受信
  PRINT D$ A$
  WAIT INP$#0(1)==CHR$(&H03)          ’EXコード待ち
  IF D$<>A$ THEN
    PRINT ”チェックサムエラー”
    PRINT#0 CHR$(&H15)                ’チェックサムエラーならNKコード出力
    GOTO *LOOP
  END_IF
  SELECT_CASE B$                      ’変数にデータを格納
    CASE "ABCDE"
      ABCDE=VAL(C$)
    CASE "FGHIJ"
      FGHIJ=VAL(C$)
    CASE_ELSE
    PRINT ”カイドクデキマセン”
  END_SELECT
  GOTO *LOOP
*CASE2
  PRINT "ABCDE=" ABCDE
  PRINT "FGHIJ=" FGHIJ
```

RS-232 のエラー処理

RS-232 で通信を行う場合にはデータの信頼性が重要になります。どんなに高度な周辺機器と接続されてもデータがデタラメでは装置として正常に稼働しません。そこで通信が正常に行われているかを常時確認する必要があるわけですが、高速で複雑な通信内容を管理するのは骨のおれることです。前記のサンプルプログラムでのチェックサムも通信内容の確認の1つで、1キャラクター受信のたびにそのチェックサムを計算してチェックサムデータと照合してデータが正常かどうかを確認しています。このようにプログラムによるチェックのほか、MPC-684ではRSE(n)関数によりRS-232の受信状態を監視してソフト上で再送信要求などのエラー処理を行うことができます。

■RSE(n)関数

n=チャンネルナンバー　0または2

返される値	内容	エラー原因として考えられるもの
0	正常終了	
1	パリティーエラー	ノイズの混入
2	オーバーラン	通信速度が速すぎる（ボーレート不一致）
4	フレーミングエラー	ケーブル不良・初期化異常
8	ブレークコード検出	異常データ

次のプログラムはMPC-684同士を接続し簡単な文字列を送受信したものですが、CNFGでの初期化に誤りがありエラーとして検出されました。エラーが発生した場合はCNFGコマンドで初期化し直します。

送信側

```
CNFG#2 "9600b8pns1NONE"                ' RS-232C CH2 初期化
 TIME 50
PRINT#2 "ABC" CHR$(&HD)
END
```

受信側

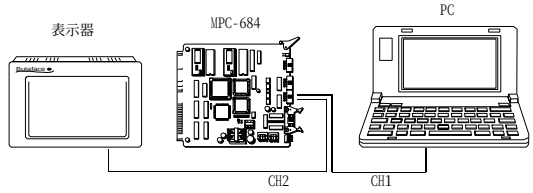
```
*LOOP
  CNFG#2 "9600b7pns1NONE"                          ' データビット長が違う
*LOOP1
  INPUT#2 A$
  SELECT_CASE RSE(2)                               ' RS-232C CH2チェック
    CASE 0 : PRINT A$ : GOTO *LOOP1
    CASE 1 : PRINT "ﾊﾟﾘﾃｨｰｴﾗｰ" : GOTO *LOOP       ' エラーが起きたら
    CASE 2 : PRINT "ｵｰﾊﾞｰﾗﾝ" : GOTO *LOOP          ' CNFGで初期化
    CASE 4 : PRINT "ﾌﾘｰﾐﾝｸﾞｴﾗｰ" : GOTO *LOOP
    CASE 8 : PRINT "ﾌﾞﾚｰｸｺｰﾄﾞ 検出"   : GOTO *LOOP
    CASE_ELSE  : PRINT "???": GOTO *LOOP
  END_SELECT
  END

  RUN
  ﾌﾚｰﾐﾝｸﾞｴﾗｰ
```

RSE(n)関数では受信時のチェックはできますが、送信時のチェックはできません。また、実際にはRSE(n)が頻繁に活躍するのでは困りものです。RS-232 のエラーに限らずトラブルは元から絶たなければ根本的な解決にはなりません。特に、別電源のパソコンや画像処理装置などの外部機器との接続にはフレームグランドの共通化による浮遊電圧の防止や配線の経路・長さ・結線の方法、そして通信フォーマットといったハード・ソフト両面での仕様を考慮して、場合によってはアイソレーターを使用するなどの対策が必要になることもあります。

タッチパネル（デジタル社製）の使用例

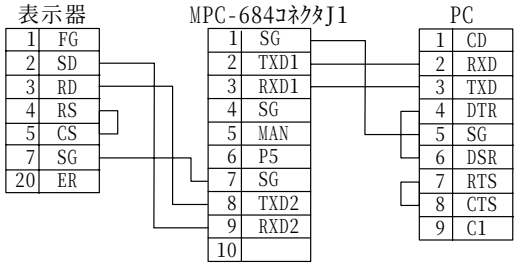
グラフィカルユーザーI/Fとしてタッチパネルが普及しています。MPC-684シリーズではMBK-SH,MBK-RSでMEWNET-FPプロトコルをサポートし、通信を意識することなくI/O機器と同レベルでGPを制御することができます。次は、RS-232通信例としてのGP制御です。
（MBK-SH,MBK-RSについては第 7 章ハードリファレンス、製品別マニュアルをご覧下さい。）
※MBK-RSは、RS-232 CH0で直接GPをサポートする機能です。

【機器接続図】



■RS-232の結線

3線式クロスで結線しました。



■RS-232の初期化

9600bps、8ビット、パリティー無し、ストップビット1、XON/OFF
MPCの初期化は CNFG#2 "9600b8pns1XON" です。表示器側の設定を確認して下さい。表示器とやりとりするデータはすべてASCII文字列です。たとえば0を送るには’0’(&H30) になります。

■MPCのRS-232送受信バッファの確認の方法

プログラムのデバッグ中にRS-232から送信したり受信したりしたデータを確認したい時があります。それにはRSコマンドを使用します。バッファの内容をASCIIコードで表示します。

```
#PRINT#2 CHR$(&H1B) "R00100001\r"
#RS 2
ＲＳ受信
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 1B<41 30 30 30 31 0D     <EC>A0001<CR>
ＲＳ送信                                         "<"記号は残バッファ先頭
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 1B 52 30 30 31 30 30 30 30 31 0D  <EC>R00100001<CR>
#
```

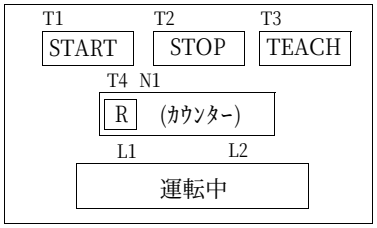
この送受信バッファは前記CNFG#2コマンドでクリアーされます。

■表示器の画面

スイッチの入力、メッセージ表示、テンキーの使用、画面切り替えを想定して2つの画面をつくりました。画面の編集はデジタル社の専用ソフトで作成しました。

【画面 1】
「START」、「STOP」、「TEACH」 のスイッチと、生産個数を表示するためのカウンターとリセットスイッチを設けてあります。その下には 「運転中」 と 「停止」 の文字を表示するようにします。

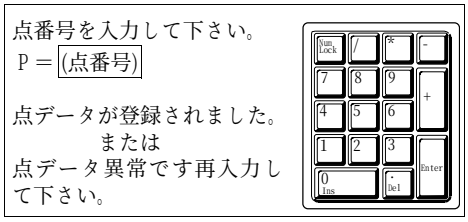
【画面 1 の構成】



RはカウンターのRESET SW

【画面 2】
点番号の入力のためのテンキーと正常な入力か否かを表示するメッセージを表示します。

【画面 2 の構成】



テンキーで入力された数字は(点番号)に表示されます(3桁)。

■動　作

- 「START」を押すとMPCは自動運転を行います。「運転中」の表示をして、カウンターの値を加算していきます。
- 「STOP」 を押すと停止します。
- 「TEACH」 を押すと表示が画面 2 に切り替わります。テンキーで点番号の設定を行います。
- 「テンキー」テンキーの数字を押すと左側に入力した数字を表示し、「ENT」キーで確定してMPCがそれを現在点とします。もしその値が1～200であれば「点データが登録されました」、それ以外なら「点データ異常です。再入力してください。」を表示します。正常な入力ならば画面 1 に戻ります。異常な入力ならば画面 2 のまま再入力になります。

■表示器システムエリアのアドレス設定

【画面１】
表示器システムエリア

SW名称	タグネーム	アドレス
「ｽﾀｰﾄ」	T1	001600
「STOP」	T2	001601
「TEACH」	T3	001602
(ｶｳﾝﾀｰﾘｾｯﾄ)	T4	001901

表示文字	タグネーム	アドレス
「運転中」	L1	001700
「停止」	L2	001701

スイッチの確認、メッセージ表示、ページ切り替えは全て表示器のシステムエリアを介して行います。スイッチが押されたらそれに対応したシステムエリアのビットが0から1となる様にします。MPCは通信でそれを読みだしてスイッチの状態を知ることができます。メッセージも割り当てられたシステムエリアのビットが０から１になると画面に表示される様にしておきます。MPCは通信でそのビットを1にセットするのです。
「START」を押すとアドレス16の第0ビットが1となります。
「STOP」を押すとアドレス16の第1ビットが1となります。
「TEACH」を押すとアドレス16の第2ビットが2となります。
(ｶｳﾝﾀｰﾘｾｯﾄ)を押すとアドレス19の第2ビットが1となります。
これらのＳＷは押されている間だけ1になります。
アドレス17の第0ビットを1にすると 「運転中」 が表示されます。
アドレス17の第1ビットを1にすると 「停止」 が表示されます。

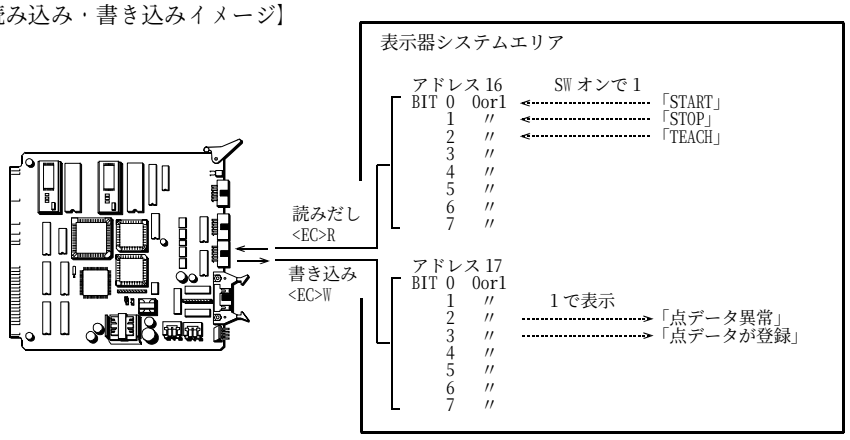
【画面２】
表示器システムエリア

SW名称	タグネーム	アドレス	
（テンキー）		001800	ライブラリーを使用。
「ＥＮＴ」	T1	001900	テンキーのＥＮＴキー

表示文字	タグネーム	アドレス	
（点番号）	K1	0020	テンキーを押すとその値が表示される
「点データ異常・・」	L1	001702	
「点データが登録・・」	L2	001703	

テンキーから入力されたデータはENTキーで確定されます。「ENT」を押すとアドレス19の第0ビットが1となります。また、そのときのデータがアドレス0020に入ります。
アドレス17の第2ビットを1にすると 「点データが登録‥」 が表示されます。
アドレス17の第3ビットを1にすると 「点データ異常‥」 が表示されます。

【読み込み・書き込みイメージ】



■画面１のSWの状態を知るには。

リードコマンドでシステムエリアからの読みだしをします。リードコマンドを送信すると対応するアドレスのデータが返ってきます。読みだしのフォーマットは次の通りです。

<EC>Raaaadddd<CR>

<EC>はESCコード(&H1B)、Rは”R”(&H52)、aaaaはアドレス、ddddは読みだすデータの数です。 MPCでは

```
PRINT#2 CHR$(&H1B) "R00100001\r"
```

を送ります。ここで注意しなければならないのがアドレスは16進数で表すことです。アドレス16(DEC)ならば10(HEX)です。返ってくるデータのフォーマットは

```
<EC>Adddd<CR>
```

です。<EC>はESCコード(&H1B)、Aは”A”(&H41)、ddddはデータです。MPCでは

```
INPUT#2 A$
```

で読み込みます。INPUT文は<CR>まで一括で読み込みます。もし「START」SWが押されていれば<EC>A0001<CR>と返ってきます。「STOP」が押されていれば<EC>A0002<CR>,「TEACH」ならば<EC>A0004<CR> です。何もおされていなければ<EC>A0000<CR>です。ここで注意しなければならないことは先頭にESCコードが入っているためこのままでは変数に変換することが出来ません。PRINT A$と実行しても表示されません。そこで次の様にして変数に変換します。

```
INPUT#2 A$                ’データをA$にいれます。
STRCPY A$ B$ 2            ’A$の3文字以降をB$にコピーします。
DATA=VAL(B$)              ’文字列B$を数字に変換します。
```

例えばこの変換で「START」が押された場合、変数DATAは1になります。

■画面１の文字表示をするには。

ライトコマンドでシステムエリアへの書き込みをします。書き込みのフォーマットは次の通りです。

```
<EC>Waaaadddd<CR>
```

<EC>はESCコード、Wは”W”(&H57)、aaaaはアドレス、ddddはデータです。「運転中」はアドレス17の第0ビットを1にすれば表示されますから、

```
PRINT#2 CHR$(&H1B) "W00110001\r"
```

とします。ここでもアドレス指定は16進数です。(DEC)17は（HEX）11です。「停止」を表示するにも同様に

```
PRINT#2 CHR$(&H1B) "W00110002\r"        です。
```

■カウントの表示

ここでは実際に何かを数えるという訳ではありませんが、一定時間毎に加算していく様にしています。このカウンターは表示器が加算してくれるのではなく、MPCから送られてくるデータを表示するだけです。

```
    I1=0
  FOR I=0 TO 9999          ’0～9999までを表示します。
    TIME 50
      TMP$=STR$(I1)
      COUNT$=""
      STRCNT=4-LEN(TMP$)       ’4byteのデータにするため”0”を連結します。
    DO WHILE STRCNT<>0         ’カウント値が”12”なら”0012”にします。
        COUNT$=COUNT$+"0"
        STRCNT=STRCNT-1
    LOOP
      COUNT$=COUNT$+TMP$
    PRINT COUNT$
     COUNT$=CHR$(&H1B)+"W0019"+COUNT$+"\r"  ’ついでにESCコードも<CR>も連結
    PRINT#2 COUNT$
      I1=I1+1
    GOSUB *COUNTER_RESET     ’カウンターリセットボタン入力確認サブルーチン
    GOSUB *READ              ’「停止」SW確認サブルーチン
    IF DATA==2 THEN : RETURN  : END_IF
  NEXT I
  RETURN
*COUNTER_RESET
  PRINT#2 CHR$(&H1B) "R00130001\r"   ’リセットボタンの入力確認
  INPUT#2 A$
  STRCPY A$ B$ 2
  IF VAL(B$)<>2 THEN : RETURN  : END_IF
    I1=0                       ’リセットが押されたらカウント変数を0クリアー
  RETURN
```

■画面の切り替え

画面の切り替えはアドレス15にページ番号を書き込みます。

```
PRINT#2 CHR$(&H1B) "W000F0002\r"    ’2ページ（画面２）へ切り替え
PRINT#2 CHR$(&H1B) "W000F0001\r"    ’1ページ（画面１）へ切り替え
```

■画面２の操作

テンキーを使うにはテンキーが設定されているアドレス18の第0ビットに1をセットしてイネーブル状態にします。

```
PRINT#2 CHR$(&H1B) "W00120001\r"
```

テンキーで入力したデータはアドレス20に入っています。MPCは「ENT」キーの押されるのを待ってこの20のデータを読み込みます。

```
(「ENT」キーの入力待ち)
DO
  PRINT#2 CHR$(&H1B) "R00130001\r"      ’アドレス19の読み込み
  INPUT#2 A$
  STRCPY A$ B$ 2
LOOP UNTIL VAL(B$)<>0
点データ入力
PRINT#2 CHR$(&H1B) "R00140001\r"        ’アドレス20の読み込み
INPUT#2 A$
STRCPY A$ B$ 2
POINT=VAL(B$)                           ’変数POINTにﾃﾝｷｰのﾃﾞｰﾀがはいります。
```

■画面２の文字表示

前記POINTの値が1～200ならば「点データが登録されました。」を表示します。それ以外の数値なら「点データ異常です。再入力して下さい。」を表示します。

```
PRINT#2 CHR$(&H1B) "W00110008\r"    ’登録
PRINT#2 CHR$(&H1B) "W00110004\r"    ’異常
```

■プログラム全体

前記プログラムをまとめたものです。(*TASK1、*TASK2はﾀﾞﾐｰです) このプログラムはスイッチの監視のために常にタスク0が通信文でLOOPしています。実際の装置ではもっと複雑な制御が要求されるので、この方法では不具合なことがあるかもしれません。表示器についているI/Oの機能を利用して、スイッチが押された時だけ通信する様に工夫すれば良いと思いますが...。

```
  FORK 1 *TASK1
  FORK 2 *TASK2
  CNFG#2 "9600b8pns1XON"
  PRINT#2 CHR$(&H1B) "W000F0001\r"
  GOSUB *CLS1
  DO
    DO
      GOSUB *READ
    LOOP UNTIL DATA<>0
    SELECT_CASE DATA
      CASE 1 : GOSUB *START
      CASE 4 : GOSUB *TEACH
    END_SELECT
    DO
      GOSUB *READ
    LOOP UNTIL DATA==0
  LOOP
*READ
  PRINT#2 CHR$(&H1B) "R00100001\r"
  INPUT#2 A$
  STRCPY A$ B$ 2
    DATA=VAL(B$)
  RETURN
*START
  GOSUB *CLS1
  PRINT#2 CHR$(&H1B) "W00110001\r"
  GOSUB *COUNT
  RETURN
*TEACH
  PRINT#2 CHR$(&H1B) "W000F0002\r"
  GOSUB *TENKEY_READ
  TIME 1000
  PRINT#2 CHR$(&H1B) "W000F0001\r"
  RETURN
*CLS1
```

```
  PRINT#2 CHR$(&H1B) "W00110000\r"
  RETURN
*COUNT
    I1=0
  FOR I=0 TO 9999
    TIME 50
      TMP$=STR$(I1)
      COUNT$=""
      STRCNT=4-LEN(TMP$)
    DO WHILE STRCNT<>0
        COUNT$=COUNT$+"0"
        STRCNT=STRCNT-1
    LOOP
      COUNT$=COUNT$+TMP$
    PRINT COUNT$
      COUNT$=CHR$(&H1B)+"W0019"+COUNT$+"\r"
    PRINT#2 COUNT$
      I1=I1+1
    GOSUB *COUNTER_RESET
    GOSUB *READ
    IF DATA==2 THEN : RETURN  : END_IF
  NEXT I
  RETURN
'
*COUNTER_RESET
  PRINT#2 CHR$(&H1B) "R00130001\r"
  INPUT#2 A$
  STRCPY A$ B$ 2
  IF VAL(B$)<>2 THEN : RETURN  : END_IF
    I1=0
  RETURN
*TENKEY_READ
  PRINT#2 CHR$(&H1B) "W00120001\r" 'TENKEY ENB
  DO
    PRINT#2 CHR$(&H1B) "R00130001\r" 'ENT KEY READ
    INPUT#2 A$
    STRCPY A$ B$ 2
  LOOP UNTIL VAL(B$)<>0
  PRINT#2 CHR$(&H1B) "R00140001\r" 'TEN KEY DATA READ
  INPUT#2 A$
  PRINT A$
  STRCPY A$ B$ 2
    POINT=VAL(B$)
  IF POINT<=0 OR POINT>200 THEN
    PRINT#2 CHR$(&H1B) "W00110004\r"
    TIME 500
    PRINT#2 CHR$(&H1B) "W00110000\r"
    GOTO *TENKEY_READ
    ELSE
    PRINT#2 CHR$(&H1B) "W00110008\r"
    TIME 500
    PRINT#2 CHR$(&H1B) "W00110000\r"
  END_IF
  RETURN
*TASK1
  DO
    FOR J=0 TO 23
      ON J
      TIME 50
      OFF J
      TIME 50
    NEXT J
  LOOP
*TASK2
  DO
    FOR K=23 TO 47
      ON K
      TIME 50
      OFF K
      TIME 50
    NEXT K
  LOOP
```