MPC-2000

Programing Tutorials

2017

株式会社アクセル

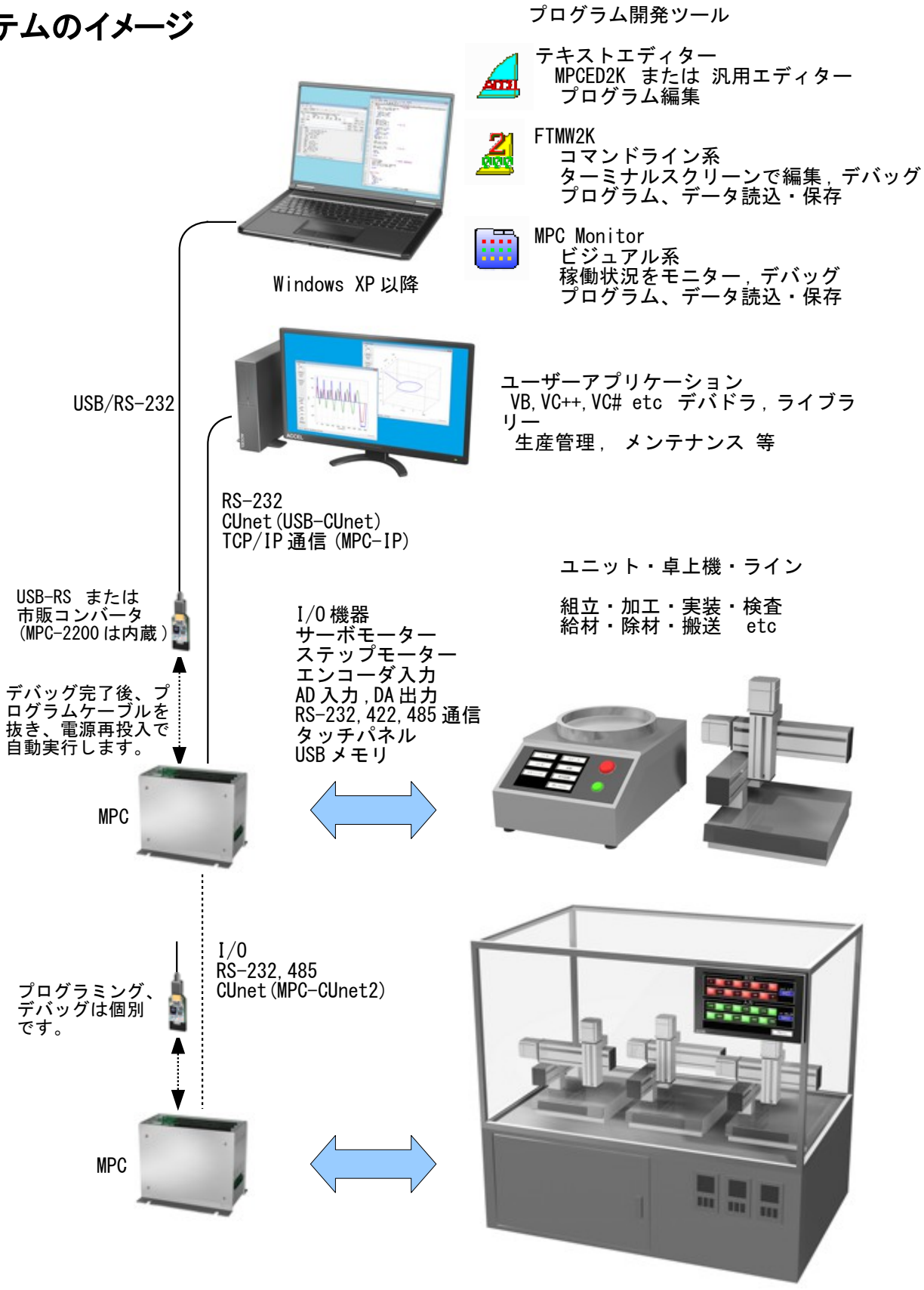
目次

開発ツールのダウンロードサイト

インストーラー、差分、サンプルなどを掲載しています。日本語版インストーラーは No.020 です。

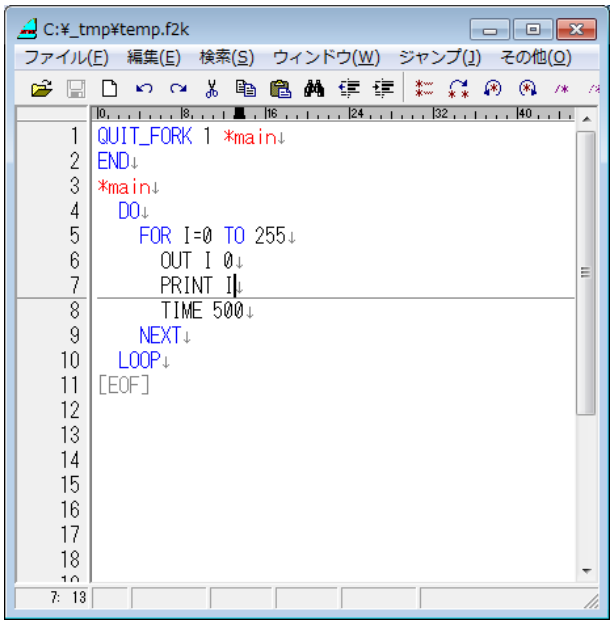
http://www.accelmpc.co.jp/japanese/ ＞「開発ツールダウンロード」

システムのイメージ



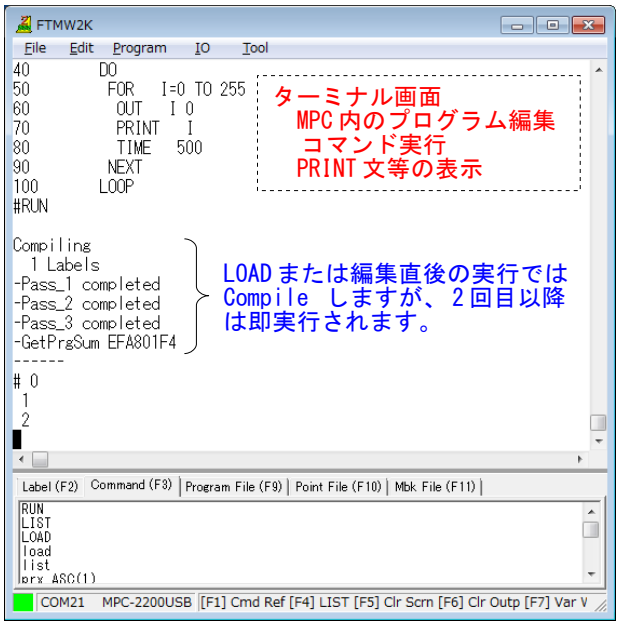
[参考資料] MPC-2000 情報サイト ＞マニュアル「FTMW2K マニュアル」、「MPC Monitor マニュアル」

プログラム開発のイメージ

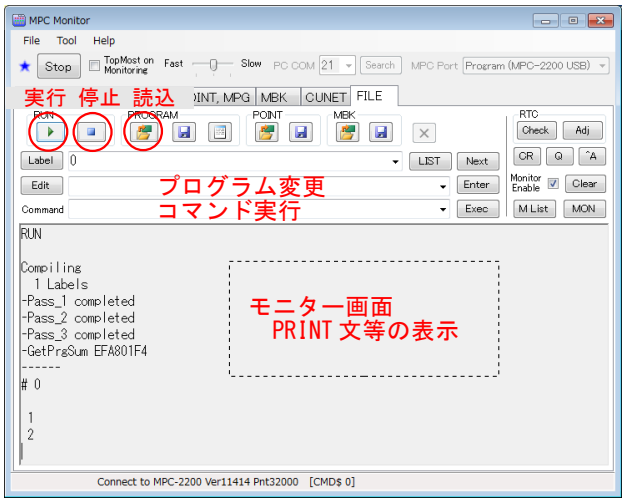


(1) エディターでコーディング

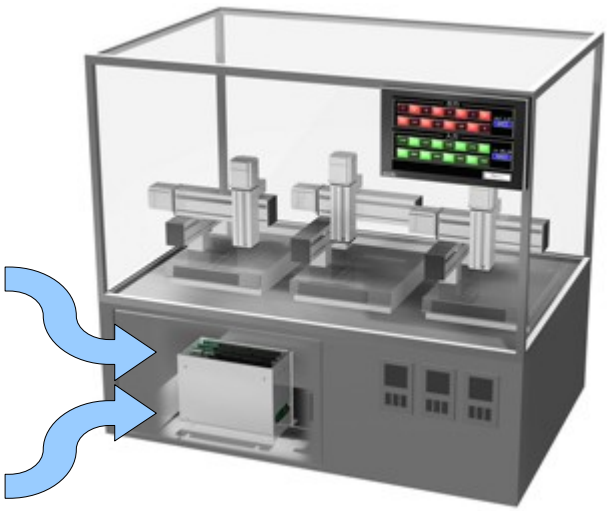
左図はMPC専用のエディタ MPCED2K ですが、使い慣れたテキストエディタをお使いください。



または



(2) FTMW2K で読込 (LOAD) して RUN 、停止は Ctrl+A



(2') MPC Monitor で MPC に読込、実行

コマンド入力

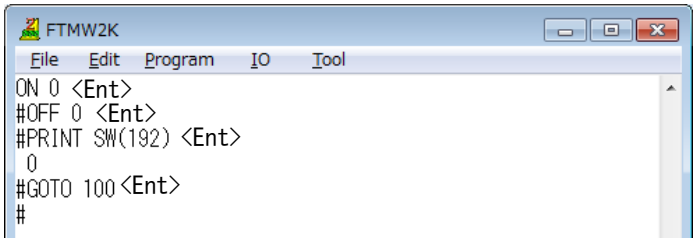
MPC はインタプリタです。プロンプトの後にコマンドを入力して Enter すると、即実行します。これをダイレクトコマンド実行といいます。

殆どのコマンドはダイレクトに実行することもプログラムに記述することもできますが、メンテナンス・編集関係などでダイレクトコマンドでしか使えないものや制御文などでプログラムにしか書けないものもあります。

両方で使える	ダイレクトのみ	プログラムのみ
ON 0	LIST	GOTO
OFF 0	MPCINIT	GOSUB
PRINT A	ERASE	IF ~
MOVL	RUN	FOR ~ NEXT
など	など	など

FTMW2K

FTMW2K のメイン画面は編集ができるターミナルスクリーンです。キー入力して Enter 押下すると MPC に送信し MPC から送られた文字を表示します。



<Ent> は PC の Enter キーです。
出力 0 オン
出力 0 オフ
入力 192 の状態調べる
GOTO はダイレクトコマンドではありません。

MPC Monitor

MPC Monitor はコマンド入力と受信文字列表示が別々です。

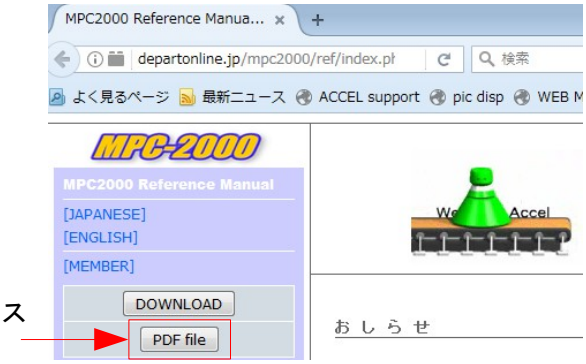
Command Combobox
に入力して
Exec
ボタンで実行します。

PDF 版コマンドリファレンス

コマンドリファレンスは「ユーザーズマニュアル」、web サイト「MPC-2000 情報」の他、PDF でも利用できます。資料持込やネットワーク接続などに制約がある作業環境で役立ちます。

「MPC-2000 情報」ページ左上

最新の PDF 版コマンドリファレンス
をダウンロードできます



初期化

MPC の初期設定や実行時のパラメータはフラッシュ ROM と S-RAM に記憶されます。開発中の試行錯誤や搬送中(特に基板単体で)の静電気印加などでパラメータが狂うと動作不良になります。次の時は初期化を励行して下さい。

- 基板単体で搬送したとき。
  搬送中にダメージを受けることがあります。
  メンテナンス等で基板単体にプログラムを入れて搬送する場合は、静電気、バッテリのショート・脱落、部品破損、結露などご注意ください。必ず帯電防止袋を使用して下さい。

- デバッグ中、挙動不審になったとき。
  あれこれやっているうちにおかしくなったとき。
  プログラムは良いはずなのにうまく動かない。(バグの可能性も追求してください)
  etc

- システムのアップデート後。

初期化コマンド

- ダイレクトコマンドで MPCINIT と ERASE を実行してください。初期化でメッセージが英語モードになります。JPN で日本語モードになります。

- 初期化するとプログラム、点データ、変数はクリアされます。必要に応じてパソコンに保存、記録を取って下さい。

```
MPCINIT
#ERASE
127*
#JPN
#
```

MPCINIT
ERASE
JPN
の順です。

バージョンの確認

FTMW2K 接続時や VER コマンドを実行するとファームウェアのバージョンを表示します。

```
VER   (1)          (2)
MPC-2000H BL/I 1.14_38 2015/07/28
 All Rights reserved. ACCEL Corp.
  PRG_430K PNT_20K DIM_20K .T32
#     (3)        (4)       (5)
```

(1) 機種
(2) バージョン
(3) プログラムエリアメモリーサイズ
(4) 点データ数 20K = 20000 点 (P(1) ～ P(20000))
(5) 配列変数容量 20K = 20000 点

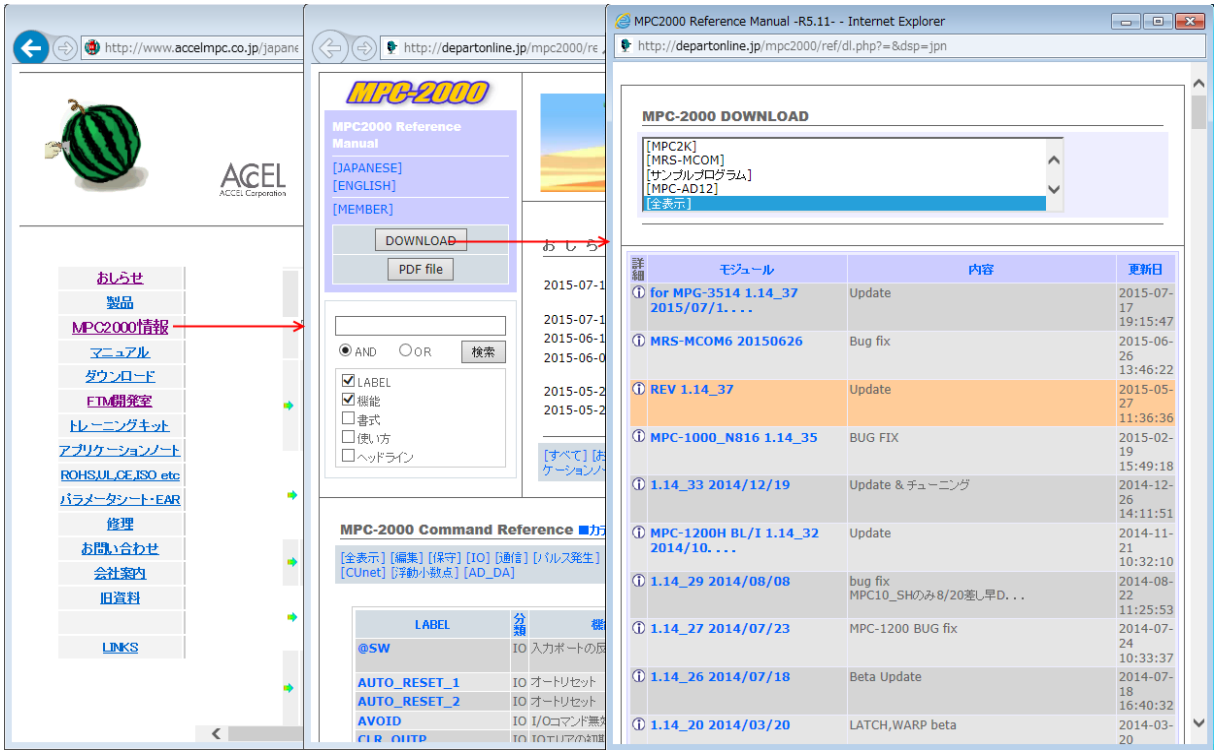
バージョンは予約文字列変数 VER$でも参照できます。

```
PRINT VER$
 1.14_38 2015/07/28
```

ファームウェアのアップデート

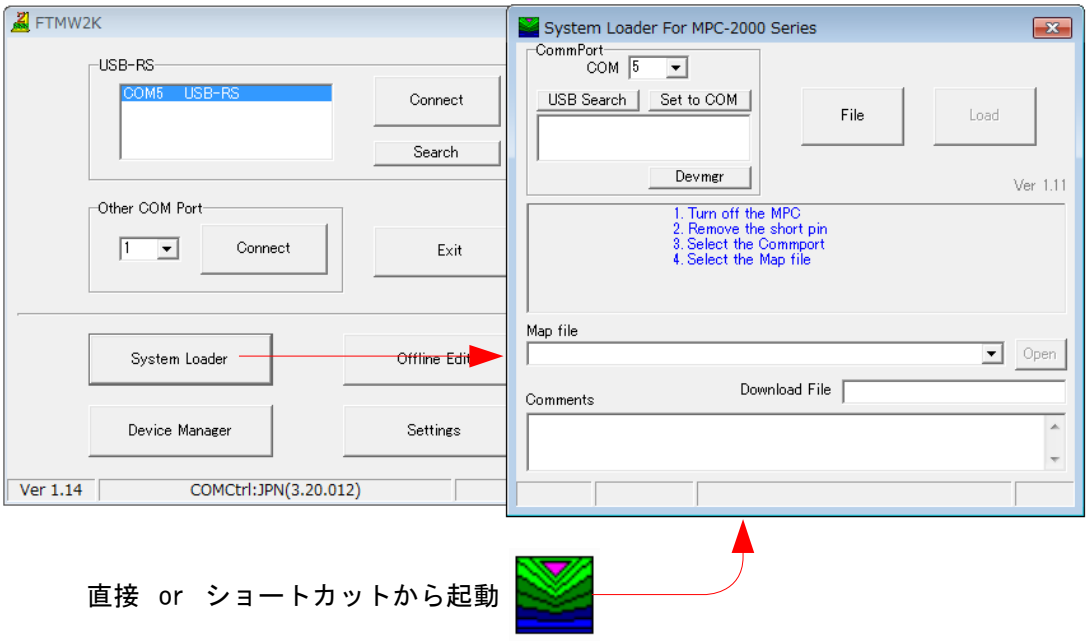
ファームウェアは MPC-2000 サイトの DOWNLOAD ページからパソコンに保存、展開してください。

http://www.accelmpc.co.jp/japanese/ → MPC-2000 → DOWNLOAD



転送ツールはシステムローダーです。MPC の電源を切り SP1 を抜いてください。
MPC-2200 は J6 コネクタ(USB Mini、SP5 はショート)、それ以外は J1 コネクタのプログラムポートにプログラミングと同じ接続です。

FTMW2K から起動　　システムローダー SYSLD2K

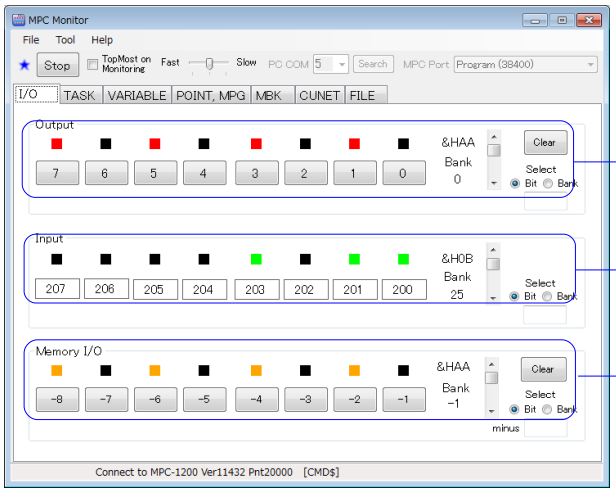


直接 or ショートカットから起動

I/O チェック

MPC Monitor

MPC Monitor は LED 表示とボタン操作です。I/O タブで実 I/O とメモリー I/O、MBK タブでタッチパネルの R エリアのチェックができます。



出力状態表示 ボタンで ON/OFF

入力状態表示

メモリー I/O 状態表示 ボタンで ON/OFF

FTMW2K

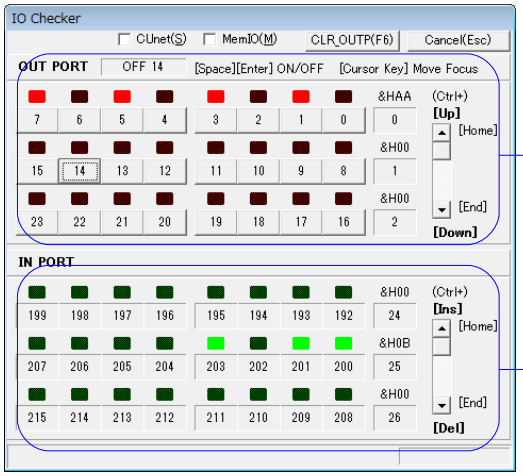
FTMW2K でチェックする場合はダイレクトコマンドで実行するか I/O チェッカーで行います。

- ダイレクトコマンド

```
ON 2                出力オン
#PRINT SW(194)      ビット入力
 1
#OFF 2              出力オフ
#PR IN(25)          8 ビット入力デシマル表示。PRINT は PR と短縮できます
 11
#PRX IN(25)         8 ビット入力ヘキサ表示
0000000B
#ON -1              メモリー I/O オン
#PR SW(-1)          メモリー I/O ビット入力
 1
#OFF -1             メモリー I/O オフ
#
```

- I/O チェッカー

メニュー IO > IO Checker (MBK エリアはできません)



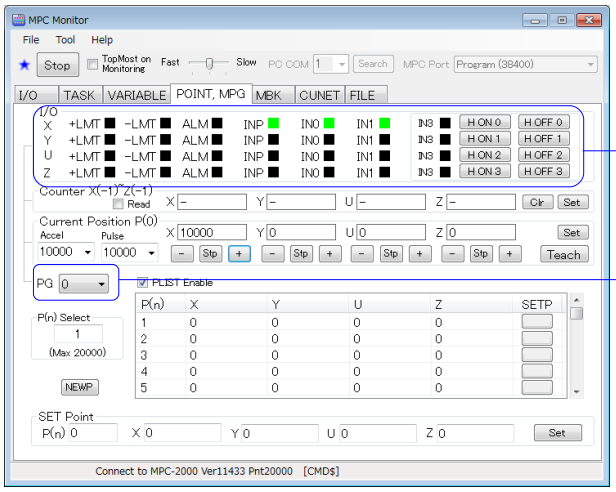
出力状態表示 ボタンで ON/OFF

入力状態表示

PG ボードの I/O チェック

MPC Monitor

POINT,MPG タブです。



PG 番号をセットすると
MPG-2314 の入力表示と出力 ON/OFF が行えます。

最初に PG 番号を指定してください。

FTMW2K

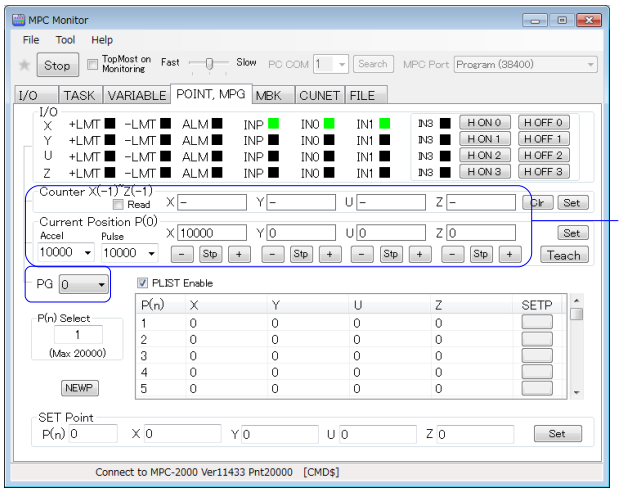
PG を指定して INCHK コマンドを実行します。

```
PG 0                                                    PG 宣言
#INCHK                                                  INCHK コマンド実行
 MPG-2314
 X=+LMT:off-LMT:off ALM:off INP:on  IN0:on  IN1:on
 Y=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
 U=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
 Z=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
#                                                       どれかのキー入力でスキャン終了
```

PG ボードのパルス出力チェック

MPC Monitor

POINT,MPG タブです。



PG(MPG-2314 は 0 ～ 9, MPC-1200 の J8 パルスは 17),
Accel(速度),Pulse(移動量)をセットして
＋ または － ボタンを押します。
ボタンの上は現在位置です。
途中停止は Stp ボタンです。

FTMW2K

- ティーチングモード

パルス出力の確認はティーチングモードが簡単です。X,Y,U,Z キー押下で指定量のパルスが出ます。

```
PG 0
#ACCEL ALL_A 10000
#T
PG=[0]@00  X=10000 Y=0 U=0 Z=0 dx=200 dy=200 du=200 dz=200
#
```

PG, ACCEL を設定します。
TEACH(省略形 T)コマンドでティーチングモードに入ります。
0, 1, 2, 3 キーで移動量の変更、
X, x, Y, y, U, u, Z, z キーでパルスを出力します。
(XYUZ:CW、xyuz:CCW)
Q, q キーでティーチングモードを終了します。

- パルスコマンドを実行

パルス発生コマンドをダイレクト実行します。

```
PG 0
#ACCEL ALL_A 10000
#RMVS X_A 1000
#RMVS X_A -1000
#
```

PG, ACCEL を設定します。
パルス発生コマンドで動作を確認します。
この場合は、絶対座標移動より相対座標移動コマンドが簡単です。

- MPG-2314 のパルスインジケータ LED

MPG-2314 は、パルス出力をボードの LED で確認できます。

プログラムとして入力するには

FTMW2K

FTMW2K のターミナル画面はプログラムの編集ができます。コマンドの前に文番号を付けて記述するとプログラムになります。Enter 押下で確定です。エラーメッセージが出たら文法に誤りが無いか確認して再入力して下さい。

```
#NEW                                    既存プログラム消去。
#10 'sample                             コメント文。
20 DO
30 PR TIME$(1)
40 TIME 500
50 LOP
50 LOP
       未定義コマンド                   間違えたらエラーメッセージ。
50 LOOP                                 再入力。

#LIST 0                                 LIST コマンドで表示。
10      'sample
20        DO
30         PRINT   TIME$(1)
40         TIME   500
50        LOOP
#RUN                                    プログラム実行。

127*
Compiling
  0 Labels
-Pass_1 completed
-Pass_2 completed
-Pass_3 completed
-GetPrgSum 718BE0BB
------
 11:19:54
 11:19:55
 11:19:55
 11:19:56
       *0    [40]                       Ctrl+A で停止。
#
```

MPC Monitor

MPC Monitor はエディタでのプログラム作成を前提としているため、直接プログラミングするには不向きです(できないことはありませんが....)。

エディターでプログラム編集

プログラムが大きくなるとFTMW2Kのターミナルでは作業が面倒になります。テキストエディターを使えばオフラインでプログラムを編集することができます。
エディターは一般のテキストエディターやWindows付属のメモ帳などが利用できます(文字コード:ANSIまたは Shift-JIS)。本文では MPC専用エディター MPCED2K.EXE を用いています。
エディターでコーディングする場合には文番号は不要です。

エディターでコーディング。文番号無。

```
'sample
DO
  PRINT TIME$(1)
  TIME 500
LOOP
```

↓ MPCへ読み込むと文番号が付けられます

```
10    'sample
20     DO
30      PRINT   TIME$(1)
40      TIME   500
50     LOOP
#
```

ラベル

GOTO,GOSUBの飛び先はラベルで指定します。文番号指定はできません。

```
*LOOP                    /* 先頭が*で始まる文字列はラベルです
  PRINT TIME$(1)
  TIME 500
  GOTO *LOOP             /* GOTO、GOSUBの飛び先はラベルで指定します
```

マルチステートメント

:(コロン)で複数のコマンドを一行に記述することができます。

```
WAIT SW(192)==1:ON 0:TIME 100:OFF 0
```

↓MPCに読み込むと整形されます。1行の長さに注意してください。

```
10      WAIT   SW(192)==1 : ON   0 : TIME   100 : OFF   0
#
```

コメント文

コメントには2種類あります。

```
'sample
DO                       ' 繰り返し
  PRINT TIME$(1)         /* 時間表示
  TIME 500
LOOP
```

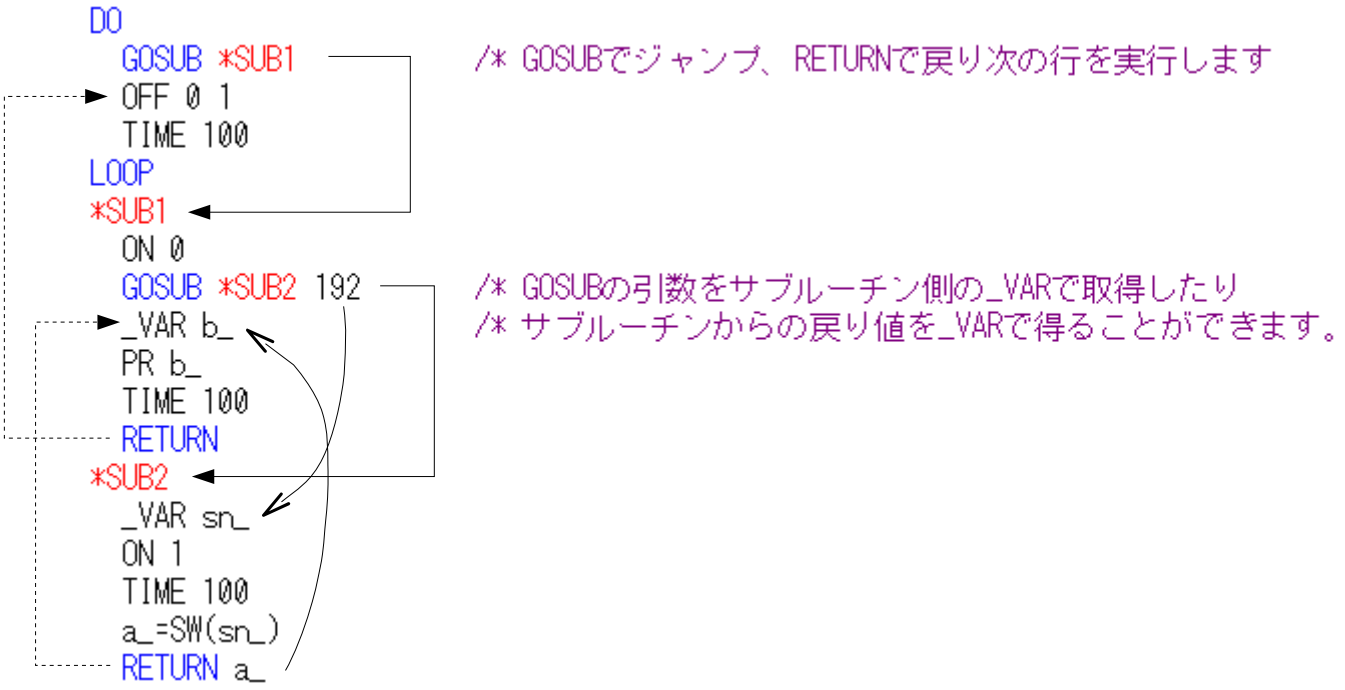
'(シンクルコーテーション)コメントはMPCに読み込まれます。
/*コメントは読み込まれません。

↓ MPCへ読込

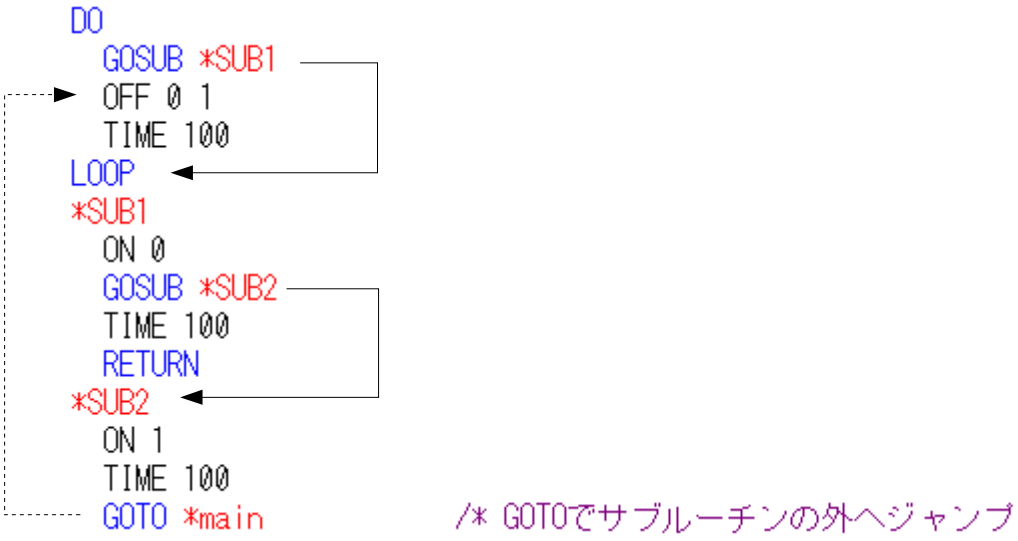
```
10    'sample
20     DO   : ' 繰り返し
30      PRINT   TIME$(1)
40      TIME   500
50     LOOP
#
```

サブルーチン

仕事の単位でサプルーチン化するとプログラムサイズのリデュース、デバッグの効率化にもつながります。



GOSUB と RETURN は必ず対応していなければなりません。



↓MPC へ読み込み、実行するとエラーになります

```
RUN

#
[50] @ 1 task     スタックが溢れました:10
```

繰り返し

GOTO で繰り返す他に、制御文として DO~LOOP、WHILE~WEND、FOR~NEXT があります。

GOTO

```
*loop
  ON 0 : TIME 100
  OFF 0 : TIME 100
GOTO *loop                  /* *loopへジャンプ
```

DO～LOOP

```
DO                          /* 無限ループ
  ON 0 : TIME 100
  OFF 0 : TIME 100
LOOP
```

WHILE~WEND

```
WHILE SW(192)==0            /* SW(192)がオフの間繰り返し
  ON 0 : TIME 100
  OFF 0 : TIME 100
WEND
```

FOR～NEXT

```
FOR C=1 TO 10               /* 回数を指定した繰り返し
  ON 0 : TIME 100           /* 途中でBREAK、GOTOで抜けることもできます
  OFF 0 : TIME 100
NEXT
PR "end" C
```

BREAK

DO～LOOP,FOR~NEXT, WHILE~WEND から抜けます。

```
C=0
DO
  ON 0 : TIME 100
  OFF 0 : TIME 100
  C=C+1
  IF C==10 THEN
    BREAK                   /* BREAKでDO~LOOPから抜けます
  END_IF                    /*   GOTOで抜けることもできます
LOOP
PR "end" C
```

条件分岐

IF 文、SELECT_CASE 文 があります。

IF

シンプルな条件判断向き

```
IF SW(192)==1 THEN      /* SW(192)がオフなら
  GOTO *MANU            /*   GOTO *MANU
ELSE                    /* そうでなければ
  GOTO *AUTO            /*   GOTO *AUTO
END_IF
*MANU
  PRINT "MANUAL"
  END
*AUTO
  PRINT "AUTO"
  END
```

SELECT_CASE

いくつもの条件があるとき。

```
DSW=IN(24)&&HF
SELECT_CASE DSW         /* DSWの値を調べる
  CASE 0                /* DSW==0なら
    PR "ABC"
  CASE 1                /* DSW==1なら
    PR "DEF"
  CASE_ELSE             /* それ以外なら
    PR "???"
END_SELECT
```

SELECT_CASE VOID

SELECT_CASE の引数を VOID とすると、CASE 文独自の論理式を評価して実行します。複数の条件を調べなければならないとき、IF 文を羅列するより効率的です。

```
SELECT_CASE VOID        /* VOIDを指定
  CASE SW(192)==1       /* SW(192)がオンなら
    ON 0
  CASE SW(193)==1       /* SW(193)がオンなら
    ON 1
  CASE SW(194)==1       /* SW(194)がオンなら
    ON 2
  CASE_ELSE
END_SELECT
```

文字列処理

代入、数値から文字列変換、連結

```
dd=25
YY$="15" : MM$="Dec" : DD$=STR$(dd)  /* 直接代入、変数を文字列変換
TODAY$=MM$+DD$+","+YY$  /* +演算子で連結ができます
PRINT TODAY$
```

```
Dec25,15
```

実行結果

```
FORMAT "00/00/00"        /* 書式を設定することができます
d=122515
TODAY$=STR$(d)           /* 10進数を文字列変換
PRINT TODAY$
```

```
12/25/15
```

実行結果

```
FORMAT "00/00/00"
d=&H122515
TODAY$=HEX$(d)           /* 16進数を文字列変換
PRINT TODAY$
```

```
12/25/15
```

実行結果

```
A$=CHR$(&H41)+CHR$(&H43)+CHR$(&H43)  /* 文字コードを文字列変換
A$=A$+CHR$(&H45)+CHR$(&H4C)
PRINT A$
```

```
ACCEL
```

実行結果

文字列から数値、コードに変換

```
TODAY$="Dec25,15"
DD=VAL(TODAY$)           /* 最初の数値を取得
YY=VAL(0)                /* 次の数値を取得
PRINT DD YY
```

```
25 15
```

実行結果

```
A$="1DE93"
A=HEX(A$)                /* HEX文字列を数値変換
PRX A : PR A
```

```
0001DE93
 122515
```

実行結果

```
TODAY$="Dec25,15"
A=ASC(TODAY$)            /* 先頭の文字のコードを取得。
PRX A
```

```
00000044
```

実行結果

文字列コピー

そのまま別の文字列変数にコピー

```
TODAY$="Dec25,15"
MerryXMas$=TODAY$        /* 文字列をそのままコピー
PRINT MerryXMas$
```

```
Dec25,15
```

実行結果

STRCPY コマンドで部分コピー

```
TODAY$="Dec25,15"
STRCPY TODAY$ MerryXMas$ 0 5  /* 0文字目(先頭)から5文字コピー
PRINT MerryXMas$
```

```
Dec25
```

実行結果

ポインタを使った部分コピー

```
TODAY$="Dec25,15"
ptr_=TODAY$              /* 予約変数ptr_に文字列先頭位置取得
ptr_=ptr_+3              /* 3文字進める
DD$=PTR$(2)              /* 2文字コピー
ptr_=ptr_+3              /* 3文字進める
YY$=PTR$(2)              /* 2文字コピー
PRINT DD$ YY$
```

```
25 15
```

実行結果

変数

MPC の変数は自動変数です。パラメータにコマンド・関数・定数・予約文字列以外の文字列を与えると変数と解釈されます。

```
VARI=0                  /* 変数に初期値を入れます
FOR I=1 TO 100          /* このIも変数です
  VARI=VARI+I
NEXT
PRINT VARI              /* PRINTは10進表示
PRX VARI                /* PRXは16進表示
```

```
# 5050
000013BA
```

実行結果

定数

CONST コマンドで文字列を定数化します。

```
CONST SOL0 0            /* 出力を定数化
CONST FEND 192          /* 入力を定数化
CONST BEND 193

ON SOL0
WAIT SW(FEND)==1        /* I/O操作を定数で表記
OFF SOL0
WAIT SW(BEND)==1
```

```
#SOL0=0
      この変数は定数化されています:20
#
```

実行結果
定数の値を変えることはできません。

配列変数

DIM コマンドで配列変数を宣言します。

```
DIM ARRY(100)           /* ARRY(0)～(99)を確保
FOR I=0 TO 99
  ARRY(I)=I             /* 配列に代入
NEXT
FOR I=0 TO 99
  PRINT "ARRY(" I ")=" ARRY(I) /* 配列の内容を表示
NEXT
```

```
# ARRY( 0 )= 0
 ARRY( 1 )= 1
 ARRY( 2 )= 2
 ARRY( 3 )= 3
```

実行結果

2 次元配列

```
DIM ARRY(3,3)                    /* 配列宣言
K=0
FOR I=0 TO 2
  FOR J=0 TO 2
    ARRY(I,J)=K : K=K+1
  NEXT
NEXT
FOR I=0 TO 2
  FOR J=0 TO 2
    PRINT "ARRY(" I "," J ")=" ARRY(I,J) /* 配列の内容を表示
  NEXT
NEXT
```

```
# ARRY( 0 , 0 )= 0
 ARRY( 0 , 1 )= 1
 ARRY( 0 , 2 )= 2
 ARRY( 1 , 0 )= 3
```

実行結果

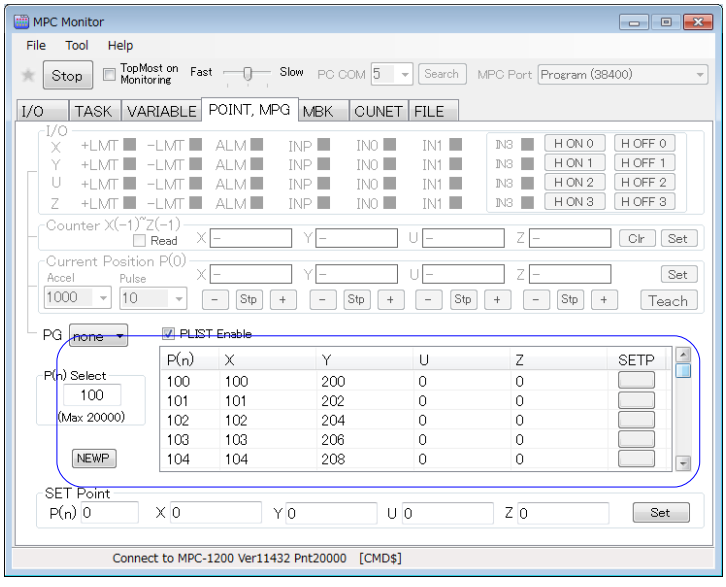
点データ(ポイントデータ)

　点データも配列変数です。DIM 配列と同等に扱えます。点データはパソコンに保存・読み込み・編集ができます。電源を切っても値を保持するバックアップ変数には点データを使用します。

```
FOR I=100 TO 200                 /* 点データのDIM宣言は要りません
  X(I)=I : Y(I)=I*2              /* X(n),Y(n),U(n),Z(n)が使えます
NEXT                             /* ※X(0),Y(0),U(0),Z(0)は不可
FOR I=100 TO 200
  PRINT "X(" I ")=" X(I) "Y(" I ")=" Y(I) /* 配列の内容を表示
NEXT
```

```
# X( 100 )= 100 Y( 100 )= 200
 X( 101 )= 101 Y( 101 )= 202
 X( 102 )= 102 Y( 102 )= 204
 X( 103 )= 103 Y( 103 )= 206
```

実行結果

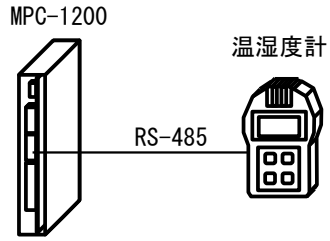


MPC Monitor の
POINT,MPG タブで
ポイントリストを参照、
変更できます。

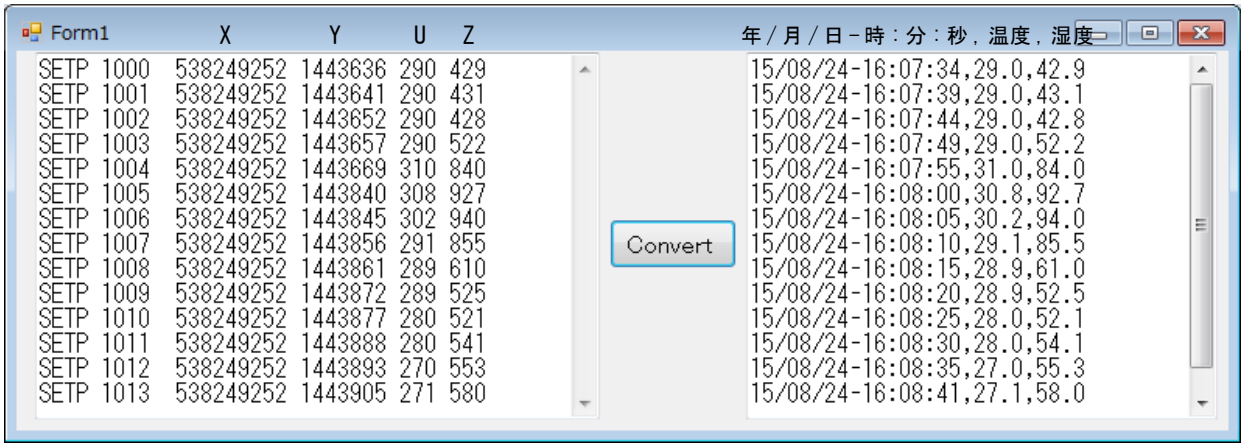
点データの利用例

　点データは四次元的な配列で、記録領域として活用できます。このサンプルは RS-485 通信の温湿度計から取得した温度湿度を MPC のクロックと共に同じインデックスの点配列に記録します。作業終了後パソコンに保存、変換して表計算ソフトで処理します。 (※記サンプル {06}=&H06,{02}=&H02,{03}=&H03)

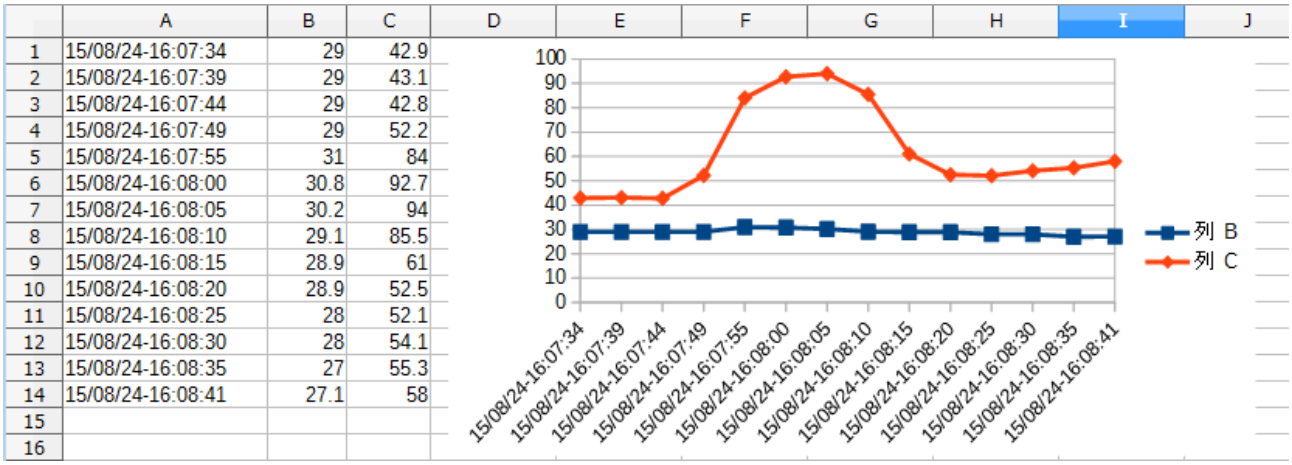
```
CNFG# 2 RS485 "9600b7pes1NONE"   /* CH2 RS-485
FILL P(1000) 1000 0              /* XYUZまとめてクリア
DO
  FOR i=1000 TO 1999
    GOSUB *GET_DATA
    X(i)=DATE(0)     /* X()に年月日 HEXで入れる 例) &H20150824
    Y(i)=TIME(0)     /* Y()に時分秒 HEXで入れる 例) &H00160734
    U(i)=temp        /* U()に温度
    Z(i)=humi        /* Z()に湿度
    TIME 5000
  NEXT
LOOP

*GET_DATA            /* 温湿度取得
PRINT# 2 CHR$(5) "02" CHR$(2) "RPV01" CHR$(3) "¥r¥n" /* 温湿度計にコマンド送信
INPUT# 2 a$          /* 温湿度計データ受信
dummy=VAL(a$)        /* 受信例 a$="[06]02[02]APV01=2001, 1,01, 0,00,0,0,0,  28.8,0,  41.8[03]"
FOR I=1 to 9         /* 数値9個読み飛ばし
  dummy=VAL(0)                                              温度      湿度
NEXT I
temp=VAL(10)         /* 温度値取得 10倍
dummy=VAL(0)         /* 読み飛ばし
humi=VAL(10)         /* 湿度値取得 10倍
PR temp humi
RETURN
```



実行してパソコンに点データを保存 (*.P2K)　→　カンマ区切りの (CSV) フォーマットに変換



↓ 表計算ソフトに読み込んでクラフ化



	A	B	C
1	15/08/24-16:07:34	29	42.9
2	15/08/24-16:07:39	29	43.1
3	15/08/24-16:07:44	29	42.8
4	15/08/24-16:07:49	29	52.2
5	15/08/24-16:07:55	31	84
6	15/08/24-16:08:00	30.8	92.7
7	15/08/24-16:08:05	30.2	94
8	15/08/24-16:08:10	29.1	85.5
9	15/08/24-16:08:15	28.9	61
10	15/08/24-16:08:20	28.9	52.5
11	15/08/24-16:08:25	28	52.1
12	15/08/24-16:08:30	28	54.1
13	15/08/24-16:08:35	27	55.3
14	15/08/24-16:08:41	27.1	58
15			
16			

ローカル変数

　ローカル変数とはタスク単位の変数で、変数名の末尾が_のものです。同じ名前でもタスク毎に違うメモリエリアに割り当てられるので、１つのサブルーチンを複数のタスクで共有することができます。

```
QUIT_FORK 1 *TASK1          /* タスク1 QUIT & FORK
QUIT_FORK 2 *TASK2          /* タスク2 QUIT & FORK
END

*TASK1                      /* タスク1
I_=0 : J_=0
DO
  GOSUB *SUM I_ J_          /* サブルーチンコール
  _VAR A_                   /* 戻り値
  PRINT "TASK" TASKn "=" A_ /* タスク番号とローカル変数の値表示
  TIME 100
  I_=I_+1 : J_=J_+1
LOOP
*TASK2                      /* タスク2
I_=0 : J_=0
DO
  GOSUB *SUM I_ J_          /* タスク1と同じサブルーチンをコール
  _VAR A_
  PRINT "TASK" TASKn "=" A_
  TIME 100
  I_=I_+2 : J_=J_+2
LOOP
*SUM                        /* 共通サブルーチン
_VAR B_ C_
SUM_=B_+C_
RETURN SUM_
```

実行結果

```
# TASK 1 = 0
 TASK 2 = 0
 TASK 1 = 2
 TASK 2 = 4
 TASK 1 = 4
 TASK 2 = 8
 TASK 1 = 6
 TASK 2 = 12
```

定数変数リスト (予約定数、予約変数)

VLIST コマンドで定数・変数のリストを表示します。

```
VLIST
■定数リスト
VOID            X_A             Y_A             U_A
Z_A             ALL_A           X_E             Y_E
U_E             Z_E             VOID_X          VOID_Y
VOID_Z          VOID_U          ALL_A           STP_I
STP_D           N_SDX           N_SDY           N_SDU
N_SDZ           P_SDX           P_SDY           P_SDU
P_SDZ           INP_ON          INP_OFF         INP_NO
ALM_ON          ALM_OFF         ALM_NO          PHASE1
PHASE2          PHASE4          UP_DWN          MD_2PLS
MD_DPLS         LMT_ON          LMT_OFF         SLMT_ON
SLMT_OFF        IN0_ON          IN0_OFF         IN1_ON
IN1_OFF         IN2_ON          IN2_OFF         IN3_ON
IN3_OFF         CW              CCW             XIN0
XIN1            XIN2            XIN3            XINP
XALM            YIN0            YIN1            YIN2
YIN3            YINP            YALM            UIN0
UIN1            UIN2            UIN3            UINP
UALM            ZIN0            ZIN1            ZIN2
ZIN3            ZINP            ZALM            SLMTp
SLMTn           LMTp            LMTn            ALM
EMG             IN0             IN1             IN2
IN3             CLR_ER          X_C             Y_C
Z_C             U_C             Lng             Wrd
Int             Ub              Lb              POS_L
NEG_L           NIL             SA0             SA1
SA2             SA3             SA4             SA5
SA6             SA7             SA8             SA9
SA10            SA11            SA12            SA13
SA14            SA15            SA0_B           SA1_B
SA2_B           SA3_B           SA4_B           SA5_B
SA6_B           SA7_B           SA8_B           SA9_B
SA10_B          SA11_B          SA12_B          SA13_B
SA14_B          SA15_B          AVOID           EOL
TMOUT           CHR_C           CLR_BUF         CMP_PLS
CMP_CNT         LONG_PRG        C_MORE          C_LESS
COM             SET_SF          RTS             RS485
USB             USB0            USB1            USB2
SACL            OUTSL           CompoWay        COMPOWAY
B7N             B7E             B7O             B8E
B8O             CAPTURE         AD7890_10       AD0
AD1             DA_TRG0         DA_TRG1         TRG0
TRG1            TRG0_END        TRG1_END        _NEXT
OFF             PGA             PGB             ON_USB
V_MIN           INC_UP          INC_LO          STR_LEN
STR_POS         EM_AS           HV_AS           SM_AS
VRING           PR_CHK          USEC            FILTER
U_W             HALF_P
Counts=187
■変数リスト
L: timer_       L: ptr_         L: rse_         L: err_
R: PG_TASK0     R: SEC          R: timer        R: MBK_ERR
R: MBK_CMD      S: VER$         R: Nm           R: Gc
R: Xn           R: Yn           R: Zn           R: An
R: Fd           R: Kn           R: In           R: Mc
R: Rn           R: MVTn         R: Jn           R: Xcp
R: Ycp          R: Zcp          R: Acp          R: Rmv
R: Xmp          R: Ymp          R: Zmp          R: Amp
R: Xpp          R: Ypp          R: Zpp          R: App
R: Fax          R: Sfx          R: Sfy          R: Sfz
R: Sfa          R: CUM_PNT      R: CUM_SRC      R: CUM_NUM
R: CUM_CNT      R: CUM_ERR      R: CUM_TASK     S: FILE$
S: FILE1$       S: FILE2$       R: MF0          R: MF1
R: MF2          R: MF3          R: MF4          R: MF5
R: MF6          R: MF7          R: CHK_SUM      R: SYSCLK
R: TASKn        R: V_PGA        R: V_PGB        R: BATTERY
R: RAM_ERR      R: COUNTER_1    R: COUNTER_2    R: AUTO_RESET_1
R: AUTO_RESET_2 R: FRQ_1        R: FRQ_2
Counts=71
      ■配列リスト

Counts=0
#
```

左は初期化後の VLIST です。
この状態で表示されるのは予約定数、予約変数です。

予約定数は主にコマンドのオプションパラメータの指定に用いられます。
変更はできません。

予約変数にはコマンド実行後の結果やシステムの情報などが入っています。
この資料にたびたび登場する SYSCLK も予約変数です。
予約変数は参照、変更ができます。

```
PRINT "SYSCLK=" SYSCLK
SYSCLK=0    /* SYSCLK リセット
TIME 1000
PRINT "SYSCLK=" SYSCLK
RUN

# SYSCLK= 24523
 SYSCLK= 1001
```

プログラムで定数、変数を使うと VLIST に追加されます。
定数を変更しようとするとエラーになります。

```
bbb=123
ccc=456
aaa=bbb+ccc
PRINT aaa
VOID=987
PRINT VOID
```

実行結果

```
# aaa= 579

[80] @ 1 task      この変数は定数化されています:20
```

```
R: AUTO_RESET_2 R: FRQ_1        R: FRQ_2        I: bbb
I: ccc          I: aaa
Counts=74
      ■配列リスト
```

演算子

```
+    加算                          A=3+2          /* A=5
-    減算                          A=3-2          /* A=1
*    乗算                          A=3*2          /* A=6
/    除算                          A=3/2          /* A=1
%    剰余                          A=3%2          /* A=1
&    論理積(AND)                   A=3&2          /* A=2
|    論理和(OR)                    A=3|2          /* A=3
^    排他的論理和(XOR)             A=3^2          /* A=1
=    代入                          A=3            /* A=3
>    比較演算子 大きい             IF A>B THEN ...
<    比較演算子 小さい             IF A<B THEN ...
!=   比較演算子 等しくない         IF A!=B THEN ...
<>   比較演算子 等しくない         IF A<>B THEN ...
==   比較演算子 等しい             IF A==B THEN ...
>=   比較演算子 以上               IF A>=B THEN ...
<=   比較演算子 以下               IF A<=B THEN ...
,    上位下位合成演算              A=(3,2)        /* A=&H00030002
;    上位 8bit 下位 24bit 合成演算 A=(3;2)        /* A=&H03000002
~    上位ワードに合成              A=3~2          /* A=&H02000003
>>   右シフト                      A=A>>1         /* 下位側に 1bit
<<   左シフト                      A=A<<1         /* 上位側に 1bit
```

- 全て 4 バイト長整数です。小数点以下切り捨てです。

- MPC-2000 シリーズの算術演算では、加減算に対して乗除算優先となり、他は左から順に演算されます。演算は式の前方から順に演算します。 優先順位が必要な場合は()で閉じてください。
  A=1+2*3
  この場合は、結果は 7 となります。1+2 を先に行う場合は、(1+2)*3 とします。

- 論理演算も同様です。
  IF IN(0)&1==IN(1)&1 THEN
  この例では、 IN(0)&1 と IN(1)&1 の比較を期待していると考えられますが、その場合は、
  IF (IN(0)&1)==(IN(1)&1) THEN
  と記述します。

- また、算術演算と論理演算の区別はありません。比較演算子は、True の時に 1、False の時に 0 という結果を出力する演算です。
  IF 文では、0 でなければ True、0 になれば False と扱います。このため、算術演算も IF 文で評価できます。たとえば IF A==1 AND SW(0)==1 THEN は IF (A==1)&(SW(0)==1) THEN となります。

- MPC-2000 は浮動小数点演算をサポートしています。

```
' Get Pie
  FLOAT   FP(6)=ACOS(SQR(3)/2)*6
  FLOAT   FP(6)=(FP(6)-3)*10
  PRINT   "PIE=3." FP(10000,6)
```

- 16 進数の記述 &H と &h の違い。
  &H と記述すると MPC の LIST では 8 文字表記されますが、&h とすれば 4 文字表記になり、1 行を短縮できます。

```
A=&HFF&&HAA              /* 大文字で記述
B=&hFF&&hAA              /* 小文字で記述
C=&hFFFFF&&hAAAAA        /* 小文字で記述しても2byte超は8文字表記
```

MPC に読み込むと

```
10      A=&H000000FF&&H000000AA
20      B=&h00FF&&h00AA
30      C=&H000FFFFF&&H000AAAAA
```

マルチタスク

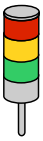
タスクは 0(メインタスク)～31 です。タスク 0 を END 終了するとプログラムを実行しならが MPC Monitor、FTMW との通信ができます。
タスク間のインターロック、データ授受はメモリー I/O や変数で行うのが一般的です。ローカル変数以外は全タスクにグローバルです。確実なインターロックのためにセマフォ関数 ON()があります。
(※タッチパネル通信(MEWNET コマンド)は上位のタスクを使用します。競合しないようにしてください。)

シンプルなマルチタスクです。*TASK1 と*TASK2 は非同期で動作します。ユニットの独立制御などに応用でます。

```
QUIT_FORK 1 *TASK1          /* タスク1 QUIT & FORK
QUIT_FORK 2 *TASK2          /* タスク2 QUIT & FORK
END
*TASK1
DO
  ON 16 : TIME 50 : OFF 16 : TIME 50    /* 出力16点滅
LOOP
*TASK2
DO
  ON 17 : TIME 100 : OFF 17 : TIME 100  /* 出力17点滅
LOOP
```

入力 192 のオン/オフでタスク 2 のルーチンを切り替えています。オート/マニュアル切替などに応用できます。(M_SW()はチャタリング防止フィルタ付きの入力)

```
QUIT_FORK 1 *main
END
*main
CLR_OUTP 1
DO
  IF M_SW(192)==1 THEN      /* トグルSW
    PR "FLICK16"
    FORK 2 *FLICK16         /* タスク2で*FLICK16を起動
    WAIT M_SW(192)==0
  ELSE
    PR "FLICK17"
    FORK 2 *FLICK17         /* タスク2で*FLICK17を起動
    WAIT M_SW(192)==1
  END_IF
  QUIT 2                    /* タスク2停止
  OFF 16 17
LOOP
*FLICK16
DO
  ON 16 : TIME 100 : OFF 16 : TIME 100  /* 出力16点滅
LOOP
*FLICK17
DO
  ON 17 : TIME 100 : OFF 17 : TIME 100  /* 出力17点滅
LOOP
```



パトライトなどを点滅するときは、そのためのタスクを起動するより PULSE_OUT コマンドが効率的です。次はどちらも SW(192)オンで点滅開始、SW(193)オンで点滅停止です。
PULSE_OUT がサポートする出力は最大 50 ポートまで。

タスクを使った点滅

```
QUIT_FORK 1 *pulseout
END
*pulseout
OFF 0 1
DO
  WAIT SW(192)==1
  QUIT_FORK 2 *led0     /* 出力0点滅タスク
  QUIT_FORK 3 *led1     /* 出力1点滅タスク
  WAIT SW(193)==1
  QUIT 2 3
  OFF 0 1
LOOP
*led0
DO
  ON 0
  TIME 500
  OFF 0
  TIME 500
LOOP
*led1
DO
  ON 1
  TIME 100
  OFF 1
  TIME 100
LOOP
```

PULSE_OUT コマンドで点滅

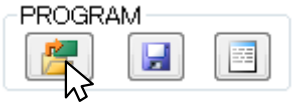
```
QUIT_FORK 1 *pulseout
END
*pulseout
OFF 0 1
DO
  WAIT SW(192)==1
  PULSE_OUT 0 10        /* 出力0を1Hzで点滅
  PULSE_OUT 1 2         /* 出力1を5Hzで点滅
  WAIT SW(193)==1
  PULSE_OUT VOID        /* 全部キャンセル
LOOP
```

[参考資料]　技術情報 「PULSE_OUT コマンドによる出力 ON/OFF」

デバッグ

MPC Monitor

MPC Monitor でのプログラム読込と基本的な動作確認方法です。
プログラムは下のように タスク 0 を END 終了してください。MPC はタスク 0 でパソコンと通信するのでタスク 0 を空けると稼動中もコマンド実行が可能になります。MPC Monitor ではこの方法が必須です。本資料のプログラムはこの方法で記述・実行しています(この部分が無いサンプルは編集で割愛しています)。



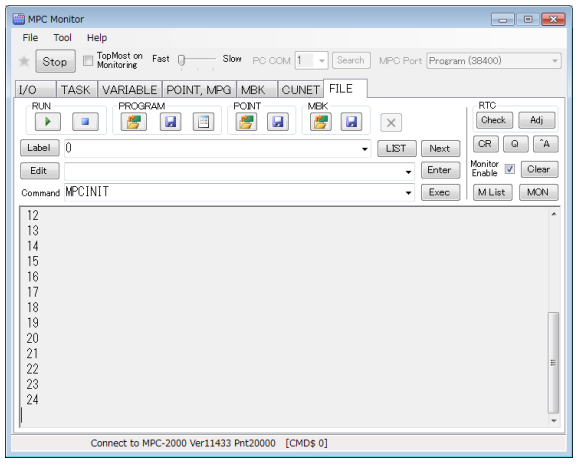
FILE タブのプログラム読込ボタンで MPC に読み込みます。

```
QUIT_FORK 1 *main
END                    /* タスク0終了
*main
I=0
DO
  WAIT M_SW(192)==1
  I=I+1
  PRINT I
  ON 0 -1
  WAIT M_SW(192)==0
  OFF 0 -1
LOOP
```

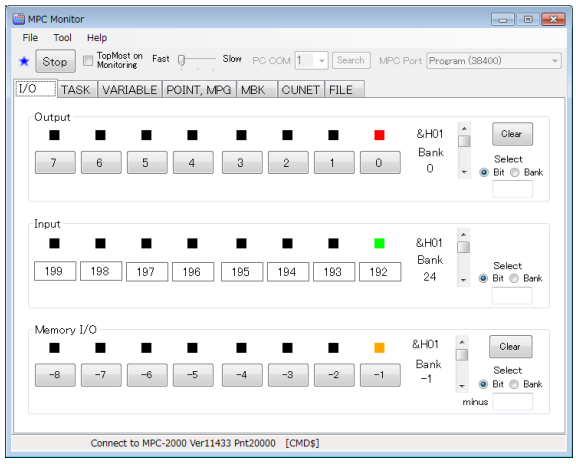
```
10      QUIT_FORK   1 *main
20      END
30    *main
40      I=0
50      DO
60        WAIT    M_SW(192)==1
70      I=I+1
80        PRINT    I
90        ON    0 -1
100       WAIT    M_SW(192)==0
110     OFF    0 -1
120   LOOP
```

プログラムを実行します。

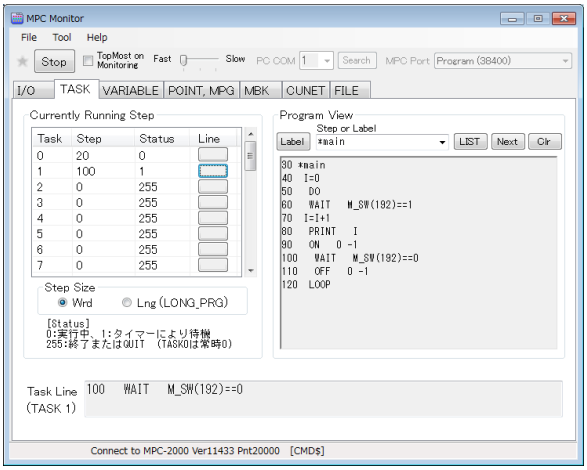
FILE タブで PRINT 文が表示されます。



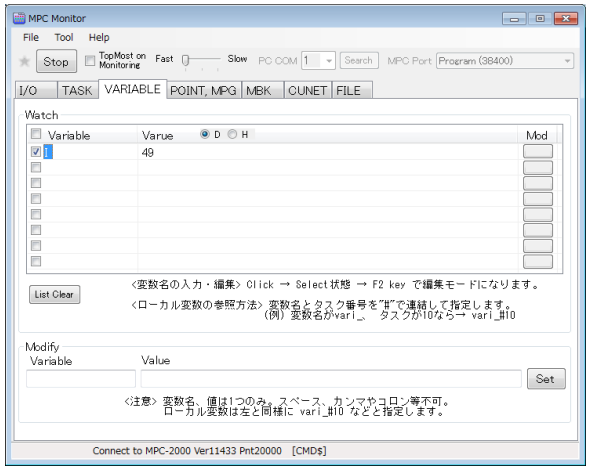
I/O タブで入出力状態がわかります。



TASK タブで実行中の文番号がわかります。



VARIABLE タブで変数の値がわかります。



FTMW2K

FTMW2K の基本的なデバッグ手法は必要箇所に PRINT コマンドを仕込んで変数や I/O 状態を表示します。
実行中のステップを知るには、MON コマンド実行、Ctrl+M 押下があります。Ctrl+A で停止したときもステップ番号が表示されます。
FTMW2K でもタスク 0 の END 終了は強力なデバッグ手法です。

```
LIST
10        QUIT_FORK   1 *main
20        END
30      *main
40       I=0
50        DO
60         WAIT   M_SW(192)==1
70       I=I+1
80         PRINT   I
90         ON   0 -1
100        WAIT   M_SW(192)==0
110        OFF   0 -1
120       LOOP
#RUN

# 1
 2
 3
MON
        *0_   [20]      *1    [60]
#
*0_     20        END
*1      60        WAIT   M_SW(192)==1

#PRINT SW(192)
 0
#       *0_   [20]      *1    [60]
#
```

タスク 0 を END で終了します。

実行

PRINT 文の出力が表示されます。

MON コマンドを実行すると、各タスクのステップ番号が表示されます。、

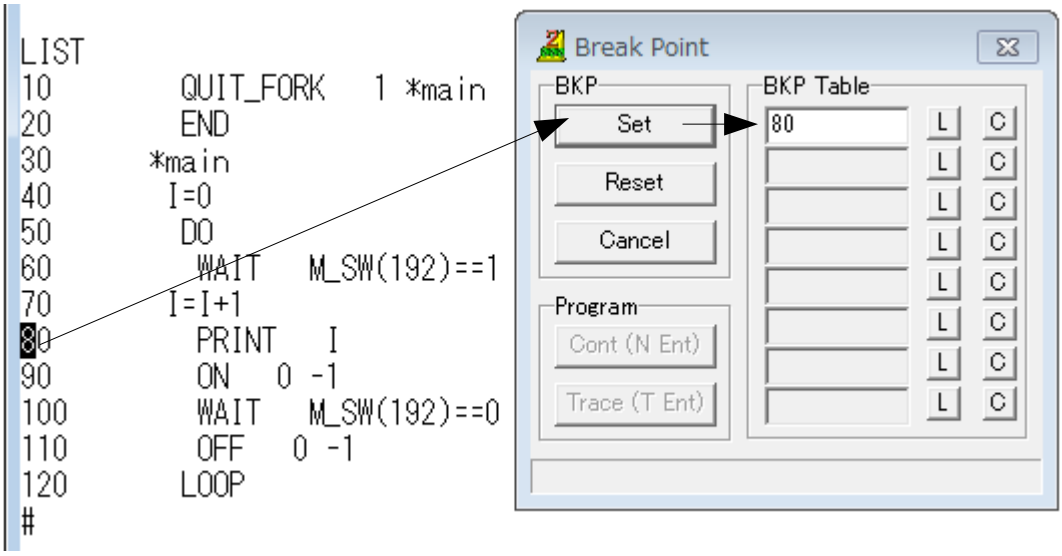
Ctrl+M で各タスクの LIST を 1 行づつ表示します。
その他のコマンドも実行可能です。

Ctrl+A で停止したときも、ステップ番号が表示されます。

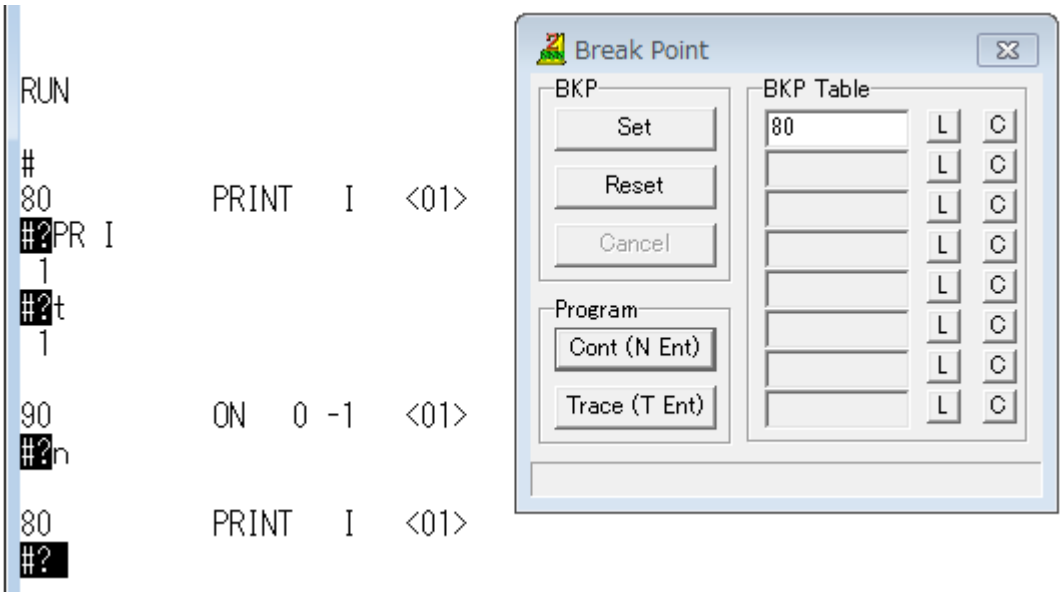
Break Point

文番号で停止箇所を指定します。全部で 8 箇所指定できます。マルチタスクプログラムの場合、タイミングがずれたりするので注意してください。
(MPC Monitor は Break Point は使えません。)

F8 で小窓が開きます。
BREAK したい行にキャレットを移動して Set ボタンを押します。



BREAK 行に到達すると #?(反転)プロンプトが表示され、PRINT コマンドなどを受け付けます。
Trace ボタンで 1 ステップ実行、Cont ボタンで再開します。Reset ボタンは前 BREAK ポイントを解除します。



サブルーチンの実行

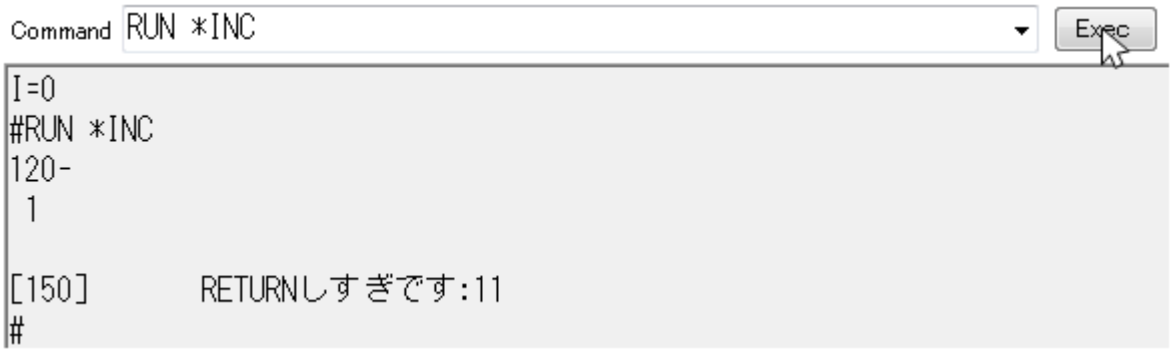
FTMW2K

RUN コマンドにサブルーチンのラベルを与えて実行します。

```
10        QUIT_FORK   1 *main
20        END
30      *main
40       I=0
50        DO
60         WAIT   M_SW(192)==1
70         GOSUB   *INC
80         ON   0 -1
90         WAIT   M_SW(192)==0
100        OFF   0 -1
110       LOOP
120     *INC
130      I=I+1
140       PRINT   I
150       RETURN
#I=0
#RUN *INC                                    サブルーチンのラベルを指定。
120-
 1

[150]   RETURNしすぎです:11                  この場合、エラーになります。
```

MPC Monitor

Command ComboBox に入力して Exec ボタンで実行です。



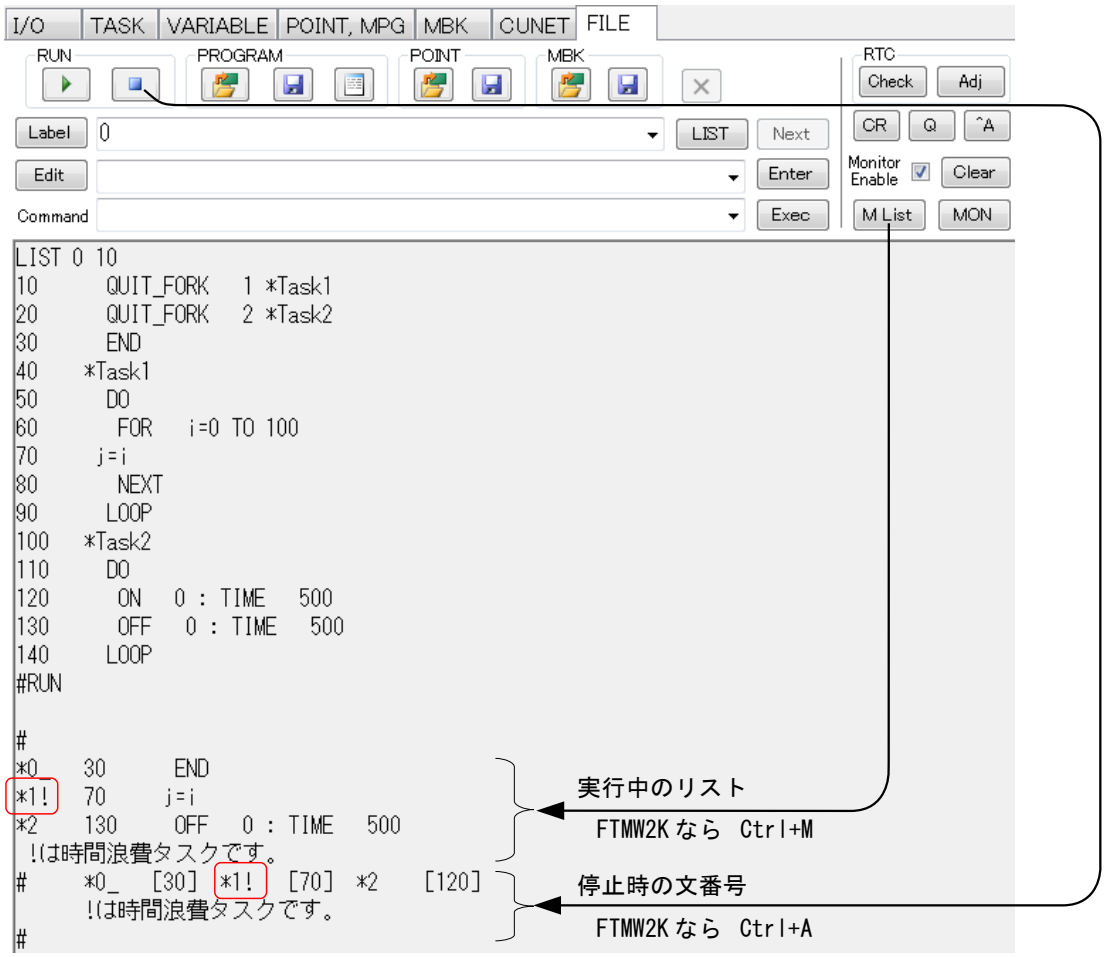
時間浪費タスクが有る場合

MPC のマルチタスクはタイムスライスのラウンドロビンです。一般的には各タスクはそれぞれの持ち時間を消費しますが、MPC は制御の効率化のため、スイッチ入力待ちや遅延などの場合、直ちに次のタスクに遷移します(タスクスワップ)。MPC のタスクスライスはデフォルトで 3msec です。

(1)のプログラムは実行中に Ctrl+M でリストを表示したり、Ctrl+A で停止したときタスク 1 に！マークが付きます。このタスクは演算のみでタスクがスワップする要素が無いため持ち時間をフルに消費します。タスク 2 は TIME コマンドが有り、その間ぼーっとしていても無駄なので即スワップします。
時間浪費タスクがあると全体のパフォーマンスが低下することがあります。

(2)は SWAP コマンドを追加しました。これにより強制的にスワップが発生し、！マークが無くなります。タスクスワップは SWAP、TIME の他、WAIT など待ち要素があるコマンドに含まれています。

(1)



(2)

```
10      QUIT_FORK   1 *Task1
20      QUIT_FORK   2 *Task2
30      END
40    *Task1
50      DO
60        FOR   i=0 TO 100
70      j=i
80        SWAP
90        NEXT
100     LOOP
110   *Task2
120     DO
130       ON   0 : TIME   500
140       OFF   0 : TIME   500
150     LOOP
#
*0_   30      END
*1    80      SWAP
*2    140     OFF   0 : TIME   500
#     *0_   [30]  *1    [80]  *2    [140]
#
```

強制スワップ

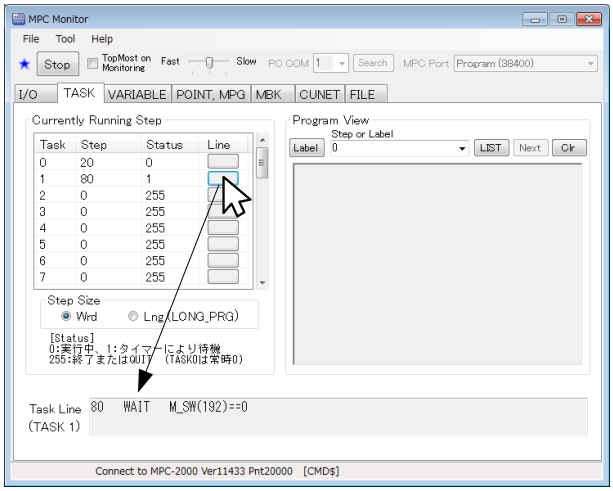
自動実行中に停止した場合

MPC Monitor は接続時に停止コードを出力しないので、装置を止めずに停止状況の確認ができます。スイッチのオンオフ待ちや変数の変化待ちなどの状況を知ることが出来ます。(タスク 0 を実行しているプログラムや MPC が通信不能状態の場合は繋がりません)

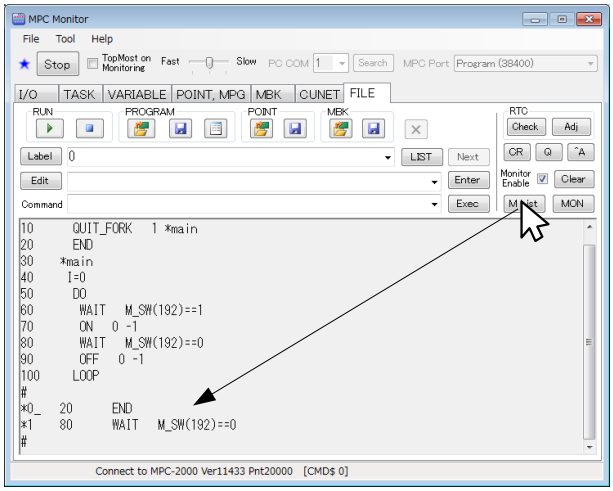
```
QUIT_FORK 1 *main
END
*main
I=0
DO
  WAIT M_SW(192)==1
  ON 0 -1
  WAIT M_SW(192)==0
  OFF 0 -1
LOOP
```

← スイッチの変化待ちで止まっていた場合。

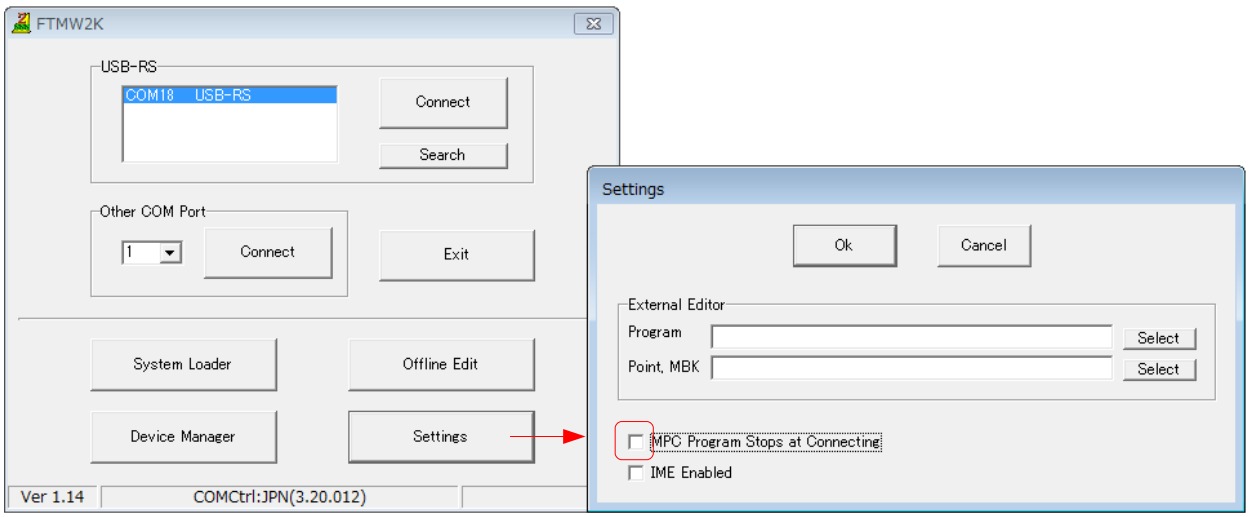
TASK タブで実行行を確認できます。
Line ボタンを押すと下にリストが表示されます。



FILE タブの M List ボタンで実行行のリストを表示します。



FTMW2K は下の設定で停止コードを出力しません。接続後、MON コマンドや Ctrl+M で停止行を調べます。タスク 0 を END 終了していないとできません。

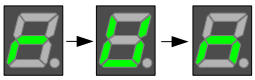


ここをアンチェック

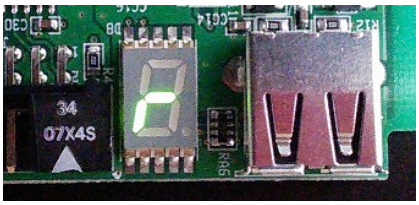
7 セグ表示内容

MPC-1200

MPC-1200 は順番に表示します。
(1.14_39 2015/09/14)



RUN ： 実行中



EDT = EDIT ： コマンド入力待ち

_ME = MEWNET ： MEWNET 起動

BLO = BATTERY LOW ： バッテリ消耗

BOU = BATTERY OUT ： バッテリ無

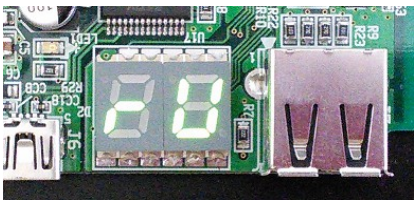
エラーコード（エラーコード表参照）
例）E10 ： スタックが溢れました

PR_LCD コマンドで表示
例）PR_LCD "ABC"

MPC-2200, MPC-2000

(1.14_38 2015/07/30)

RU = RUN ： 実行中



WA = WAIT ： コマンド入力待ち
（この場合 "_" は "W"）

_M = MEWNET ： MEWNET 起動

BL = BATTERY LOW ： バッテリ消耗

BO = BATTERY OUT ： バッテリ無

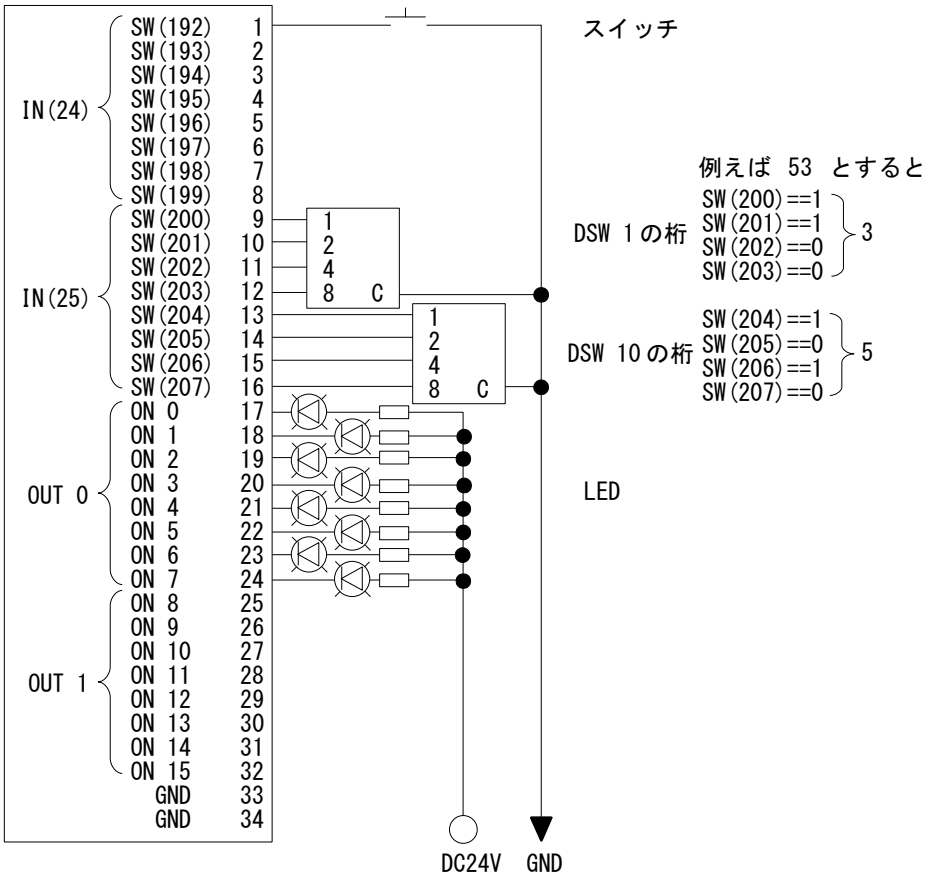
エラーコード（エラーコード表参照）
例）（ドット）10 ： スタックが溢れました

PR_LCD コマンドで表示
例）PR_LCD "AB"

I/O 制御

I/O はビット単位、またはバンクで扱うことができます。バンクの基本は 1 byte 長ですが~Wrd,~Lng でキャストすると 2byte 長、4Byte 長で扱えます。
ビットの場合は 0(OFF)か 1(ON)、バンクでは 0～&HFF(~Lng なら&HFFFFFFFF)の値になります。
扱える I/O は、実 I/O・メモリー I/O・タッチパネル I/O・CUnet I/O です(タッチパネル I/O は 2Byte 長が基本です)。

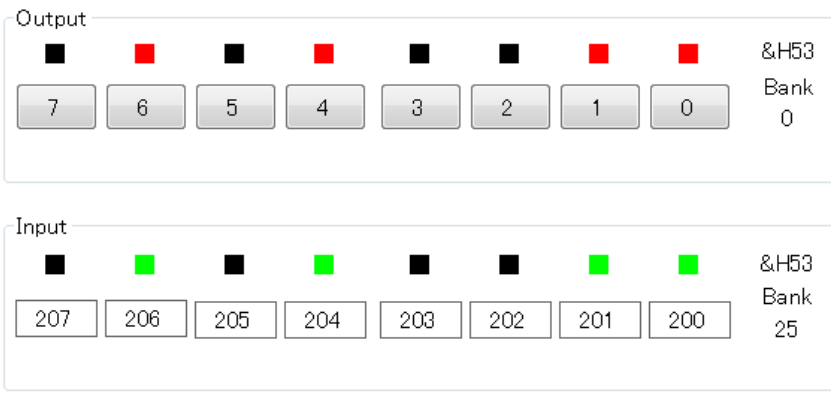
次はデジスイッチを 2 桁接続して 10 進数にしています。




```
WAIT SW(192)==0          /* ビット入力　SW(192)オフ待ち
WAIT SW(192)==1          /* ビット入力　SW(192)オン待ち
DswHex=IN(25)            /* 8ビットパラレル入力
DswHi=(DswHex>>4)*10     /* 10の桁。4ビット右シフトして10倍
DswLo=DswHex&&HF         /* 1の桁。下位4ビット
DswDec=DswHi+DswLo       /* 足す
PRINT DswDec             /* 表示
OUT DswHex 0             /* 8ビットパラレル出力
```

```
53
```

実行結果



MPC Monitor の
I/O タブでみると

パルス発生

主なコマンド

PG コマンド
　そのタスクがアクセスする MPG をアサインします。MPG-2314 は PG 0～9、MPC-1200 は PG 17 です。別々のタスクで同一 PG を指定したり、同じタスクで PG を切り替えることも可能です。
ACCEL コマンド
　最高速度、加減速と起動速度を設定します。基本は台形駆動ですが S 字加減速もできます。
FEED コマンド
　ACCEL で設定した最低速と最高速の範囲で 0~100 で速度を設定します。通常、ACCEL は初期設定とし、稼働中に速度を変更する場合は FEED を用います。
CLRPOS コマンド
　現在位置を 0 にします。任意の値に設定するには SETP コマンドまたは STPS コマンドを用います。
RMVL,RMVS コマンド
　相対座標移動のパルス発生です。RMVL は直線補間移動(MPG-2314)。
MOVL,MOVS コマンド
　絶対座標移動のパルス発生です。MOVL は直線補間移動(MPG-2314)。
RR()関数
　パルス発生が完了するのを待ちます。基本的にパルス発生コマンドは実行後次のステップに進むので、RR()で完了するのを待ったり、パルス発生中の処理などを行います。
軸指定定数
　上記のようにコマンドに X_A と与えると X 軸に対して作用します。ALL_A は全軸、X_A|U_A というように論理和をとると X 軸と U 軸に作用します。MPG-2314 で MOVL に X_A|U_A 指定すると X と U が直線補間になります。RMVS,RMVL は論理和はできません。

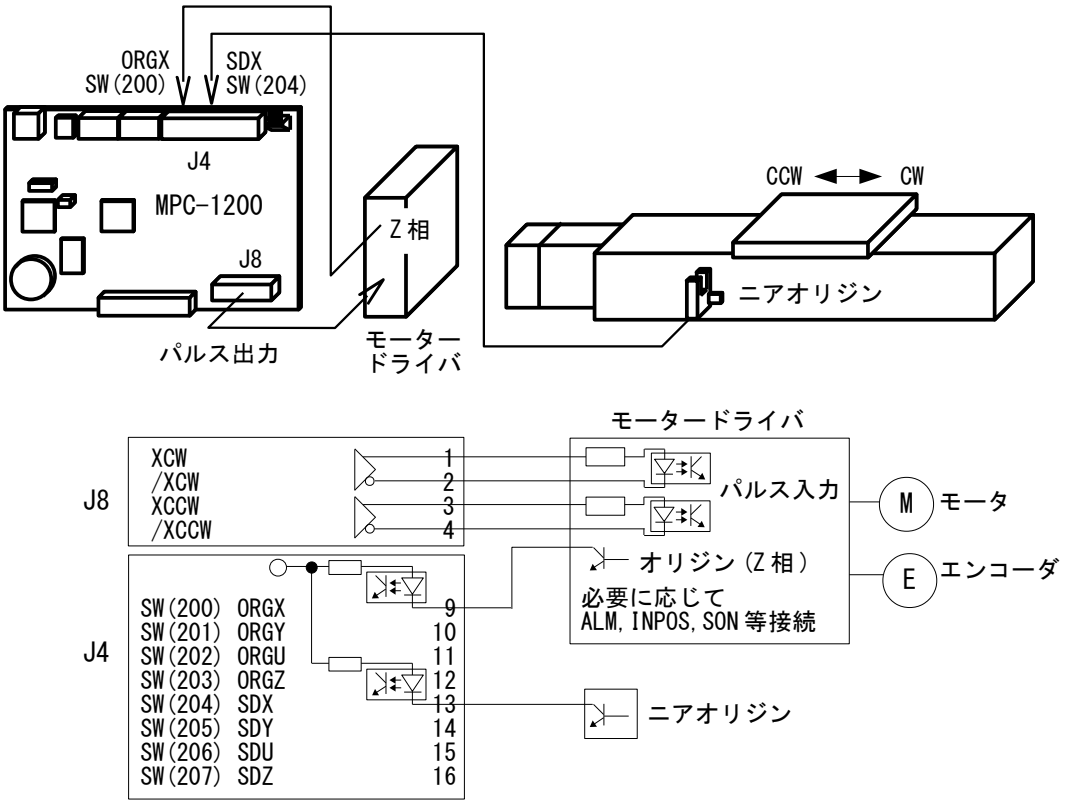
パルス発生例

　マルチタスクで MPC-1200 と MPG-2314 のパルス発生を同時に行います。MPC-1200 は絶対座標移動、MPG-2314 は相対座標移動ですが動作は同じです。この場合、直線補間にはなりません。

```
QUIT_FORK 1 *MPC1200
QUIT_FORK 2 *MPG2314
END                             /* Task0(メインタスク)終了
*MPC1200                        /* MPC-1200タスク
PG 17                           /* MPC-1200のJ8からパルス発生
ACCEL X_A|Y_A 50000 5000 500    /* 最高速、加減速設定
FEED X_A|Y_A 100
CLRPOS X_A|Y_A                  /* 現在位置クリア
DO
  MOVS 10000 5000               /* XY軸絶対座標移動
  WAIT RR(X_A|Y_A)==0           /* 移動完了待ち
  TIME 500
  MOVS 0 0                      /* XY軸絶対座標移動
  WAIT RR(X_A|Y_A)==0           /* 移動完了待ち
  TIME 500
LOOP
*MPG2314                        /* MPG-2314タスク
PG 0                            /* MPG-2314のJ1からパルス発生
ACCEL X_A|Y_A 50000 5000 500    /* 最高速、加減速設定
FEED X_A|Y_A 100
CLRPOS X_A|Y_A                  /* 現在位置クリア
DO
  RMVS 10000 5000               /* XY軸相対座標移動
  WAIT RR(X_A|Y_A)==0           /* 移動完了待ち
  TIME 500
  RMVS -10000 -5000             /* XY軸相対座標移動
  WAIT RR(X_A|Y_A)==0           /* 移動完了待ち
  TIME 500
LOOP
```

原点復帰

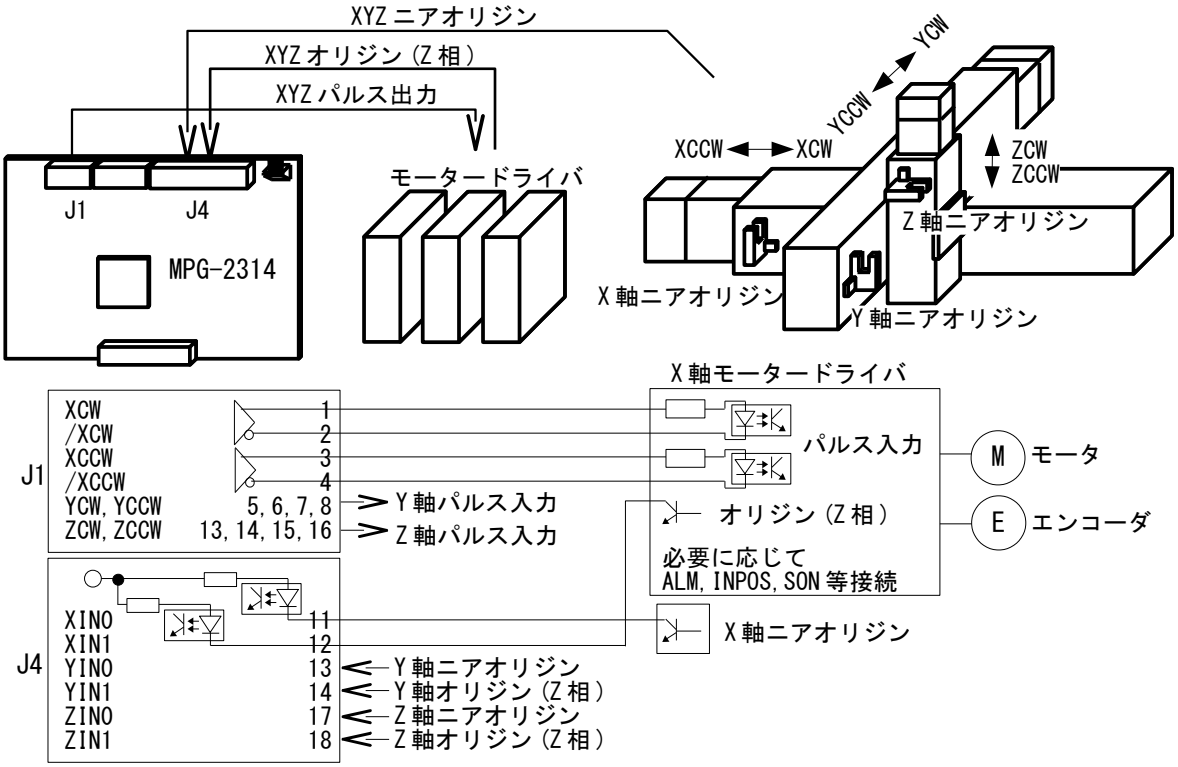
MPC-1200 の単軸原点復帰例




```
PG 17                          /* PGアサイン
ACCEL X_A 20000 2000 500       /* SDXからORGXまでは500ppsで動作
IF SW(204)==1 THEN             /* ニアオリジンが入っていたら
  FEED X_A 100
  RMVS X_A 10000               /* 軸退避移動
  WAIT RR(X_A)==0              /* 退避移動完了待ち
END_IF
FEED X_A 25                    /* 必要に応じて
SHOM &H54                      /* SDXを有効にする
HOME X_A NEG_L                 /* 原点復帰 SDXまでACCELのmax, ORGXまでmin
WAIT RR(X_A)==0                /* 原点復帰終了待ち
CLRPOS X_A                     /* 現在点クリア
```

MPG-2314 の 3 軸原点復帰

最初に Z 軸を上昇させてから XY 軸が同時に原点復帰します。



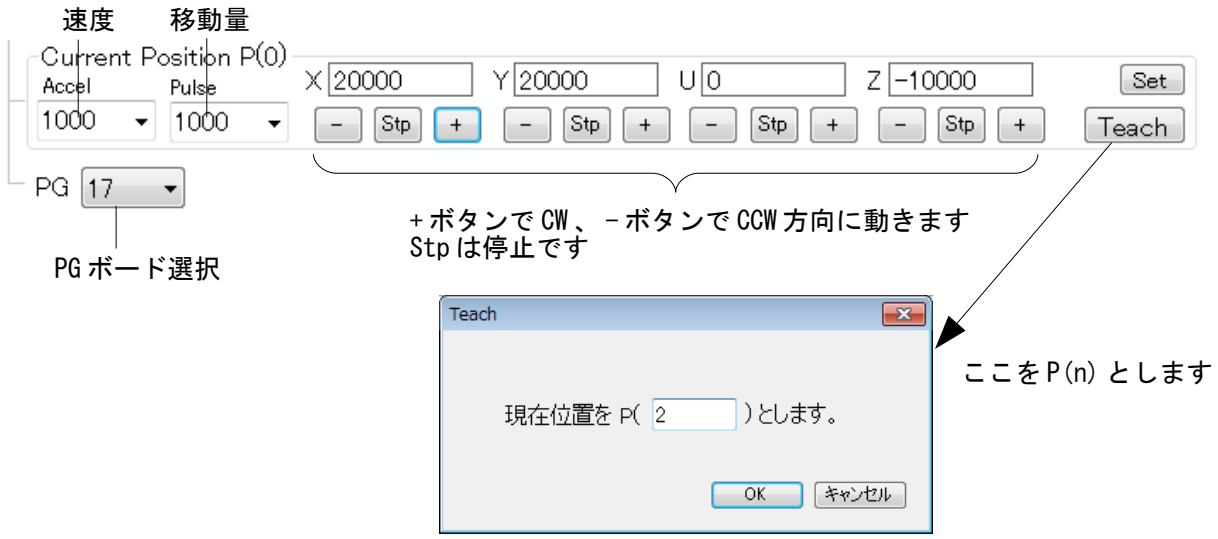

```
PG 0                          /* PGアサイン MPG-2314 DSW1=0
ACCEL X_A|Y_A|Z_A 20000 1000 5000   /* ニアオリジンONまで20000pps
                                    /*  それからオリジンONまで5000pps
IF HPT(ZIN0)==1 THEN          /* Z軸ニアオリジンがONなら退避移動
  RMVS 0 0 0 -20000
  WAIT RR(ALL_A)==0
END_IF
SHOM Z_A IN0_ON|IN1_ON|CW /* Z軸ニアオリジンON→オリジンON CW方向
HOME Z_A POS_L                /* Z軸原点復帰
WAIT RR(Z_A)==0
IF HPT(XIN0)==1 THEN          /* X軸ニアオリジンがONなら退避移動
  RMVS 20000 0 0 0
  WAIT RR(ALL_A)==0
END_IF
IF HPT(YIN0)==1 THEN          /* Y軸ニアオリジンがONなら退避移動
  RMVS 0 20000 0 0
  WAIT RR(ALL_A)==0
END_IF
SHOM X_A|Y_A IN0_ON|IN1_ON|CCW  /* XY軸ニアオリジンON→オリジンON CCW方向
HOME NEG_L NEG_L              /* XY軸原点復帰
WAIT RR(X_A)==0
CLRPOS
```

ティーチングによる点データの作成

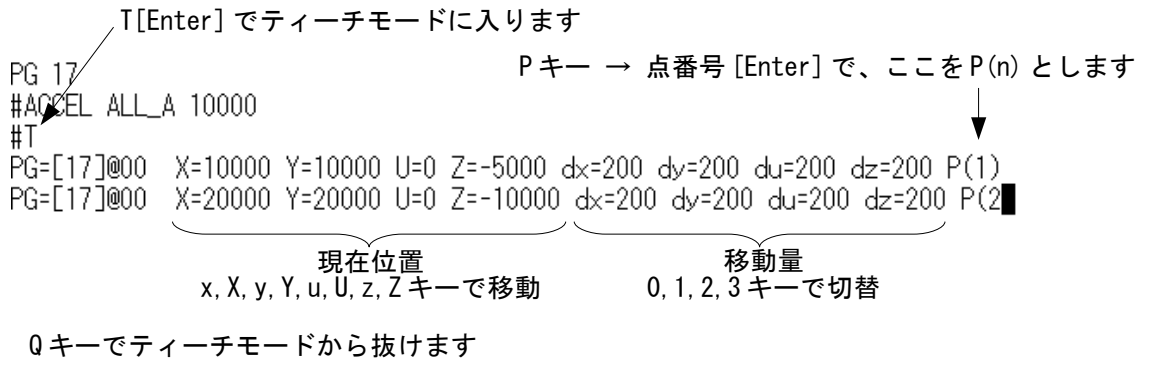
　目的の位置まで Jog 移動して点として教示します。このパルス発生は配線後の動作確認にも役立ちます。
※実装置では、タッチパネルにるティーチング機能が実用的です。

MPC Monitor

　POINT,MPG タブ

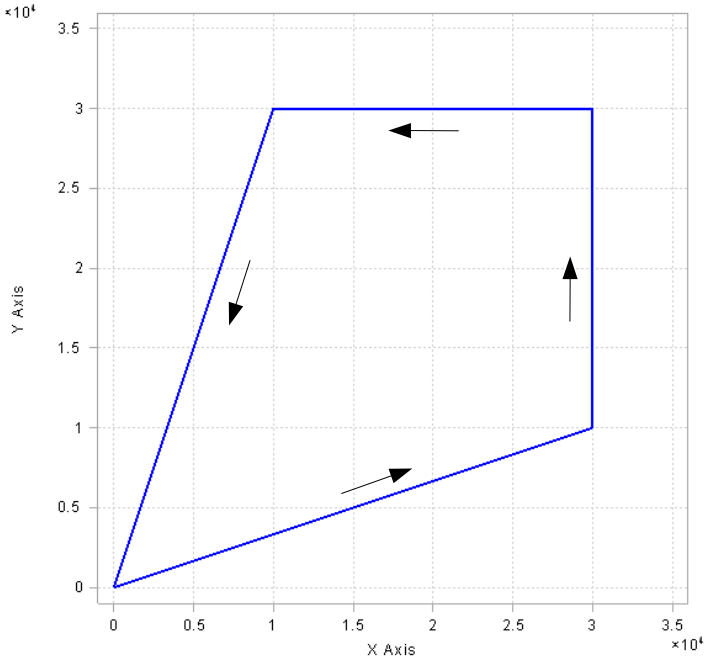


FTMW2K



移動例

　MPC には特定の位置を指定して移動する絶対座標移動と、現在位置から移動量を指定する相対座標移動があります。座標、移動量は定数、変数、点データで指定することができます。
下記の４例は同じ動作です。



絶対座標移動

```
PG 0                              /* MPG-2314 #0
ACCEL X_A|Y_A 10000 1000 500      /* XY軸 最高速、加減速設定
CLRPOS X_A|Y_A                    /* XY軸 現在位置を0にする
MOVL 30000 10000                  /* XY軸 絶対座標30000,10000へ移動
WAIT RR(X_A|Y_A)==0               /* XY軸 パルス発生終了待ち
MOVL 30000 30000                  /* XY軸 絶対座標30000,30000へ移動
WAIT RR(X_A|Y_A)==0
MOVL 10000 30000                  /* XY軸 絶対座標10000,30000へ移動
WAIT RR(X_A|Y_A)==0
MOVL 0 0                          /* XY軸 絶対座標0,0へ移動
WAIT RR(X_A|Y_A)==0
```

相対座標移動

```
PG 0                              /* MPG-2314 #0
ACCEL X_A|Y_A 10000 1000 500      /* XY軸 最高速、加減速設定
CLRPOS X_A|Y_A                    /* XY軸 現在位置を0にする
RMVL 30000 10000                  /* XY軸 +30000,+10000パルス移動
WAIT RR(X_A|Y_A)==0               /* XY軸 パルス発生終了待ち
RMVL 0 20000                      /* XY軸 0,+20000パルス移動
WAIT RR(X_A|Y_A)==0
RMVL -20000 0                     /* XY軸 -20000,0パルス移動
WAIT RR(X_A|Y_A)==0
RMVL -10000 -30000                /* XY軸 -100000,-30000パルス移動
WAIT RR(X_A|Y_A)==0
```

座標を変数で指定

パルス発生部分をサブルーチンにして、パラメータ(座標値)を変数で受け取ります。

```
  PG 0                                /* MPG-2314 #0
  ACCEL X_A|Y_A 10000 1000 500        /* XY軸 最高速、加減速設定
  CLRPOS X_A|Y_A                      /* XY軸 現在位置を0にする
  GOSUB *MOVE 30000 10000             /* サブルーチンのパラメータに座標値を与える
  GOSUB *MOVE 30000 30000
  GOSUB *MOVE 10000 30000
  GOSUB *MOVE 0 0
  END
*MOVE
  _VAR x y
  MOVL x y                            /* 座標 x y に移動
  WAIT RR(X_A|Y_A)==0
  RETURN
```

点に移動

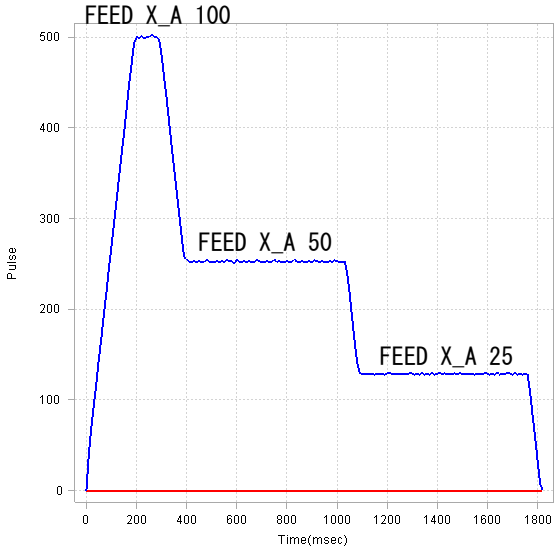
SETP コマンドで点を作成し、そこへ順番に移動します。

```
SETP 1000 30000 10000 0 0       /* P(1000)
SETP 1001 30000 30000 0 0       /* P(1001)
SETP 1002 10000 30000 0 0       /* P(1002)
SETP 1003 0 0 0 0               /* P(1003)
PG 0                            /* MPG-2314 #0
ACCEL ALL_A 10000 1000 500      /* 最高速、加減速設定
CLRPOS  ALL_A                   /* 全軸 現在位置を0にする
FOR p=1000 TO 1003
  MOVL P(p)                     /* 点に移動
  WAIT RR(ALL_A)==0             /* パルス発生終了待ち
NEXT
```

マルチタスク例

　パルス発生コマンドは実行後次のステップに進みます(HOME,JUMP,WARP を除く)。そのため同一タスクでパルス発生中の処理ができます。この例では*PG タスクでパルスを出し、その数を監視して速度を変化させています。
　また、別のタスクからも同じ PG にアクセスできます。この例では*PgRecord タスクでパルスの出力状況を取得して点データ配列に記録します。右図は作業終了後、その記録をパソコンにアップして速度変化をグラフにしたものです。

```
QUIT_FORK 1 *PG
END                          /* Task0(メインタスク)終了
*PG
MEWNET 38400 1
QUIT_FORK 2 *PgRecord
PG 0                         /* MPG-2314 #0
ACCEL X_A 50000 5000 500     /* 最高速、加減速設定
CLRPOS X_A                   /* 現在位置クリア
FEED X_A 100                 /* 速度100%
MOVS X_A 40000               /* X軸移動
WAIT X(0)>10000              /* 10000超待ち
FEED X_A 50                  /* 速度50%
WAIT X(0)>30000              /* 30000超待ち
FEED X_A 25                  /* 速度25%
WAIT RR(X_A)==0              /* 移動完了待ち
TIME 20
QUIT 2
MOVS X_A 0
WAIT RR(X_A)==0
END

*PgRecord                    /* 記録タスク
PG 0
FILL P(10000) 10000 0
oldposX_=X(0)
oldposY_=Y(0)
oldposZ_=Z(0)
pointnum_=10000              /* 加減速記録先頭点番号
locusnum_=15000              /* 軌跡記録先頭点番号
SYSCLK=0
interval_=10                 /* 記録間隔msec
DO
  WAIT SYSCLK%interval_==0
  nowclk_=SYSCLK
  nowposX_=X(0)              /* X現在位置
  nowposY_=Y(0)              /* Y現在位置
  nowposZ_=Z(0)              /* Y現在位置
  difposX_=nowposX_-oldposX_  /* X差分
  oldposX_=nowposX_
  difposY_=nowposY_-oldposY_  /* Y差分
  oldposY_=nowposY_
  difposZ_=nowposZ_-oldposZ_  /* Z差分
  oldposZ_=nowposZ_
  /*   点番号    時間     X差分    Y差分     Z差分
  SETP pointnum_ nowclk_ difposX_ difposY_ difposZ_
  pointnum_=pointnum_+1
  /*   点番号    時間    X現位置   Y現位置   Z現位置
  SETP locusnum_ nowclk_ nowposX_ nowposY_ nowposZ_
  locusnum_=locusnum_+1
  WAIT SYSCLK%interval_<>0
LOOP
```

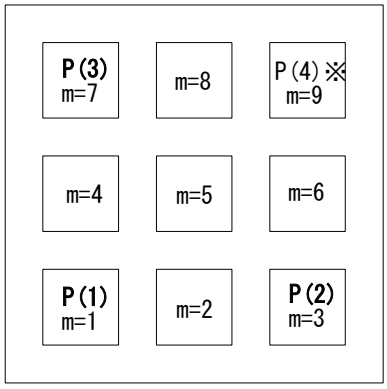


縦軸は 10msec 単位のパルス数 ∴ 500=50000pps
インタプリタ、マルチタスクの影響でジッタが出ていますが、実際は滑らかです。

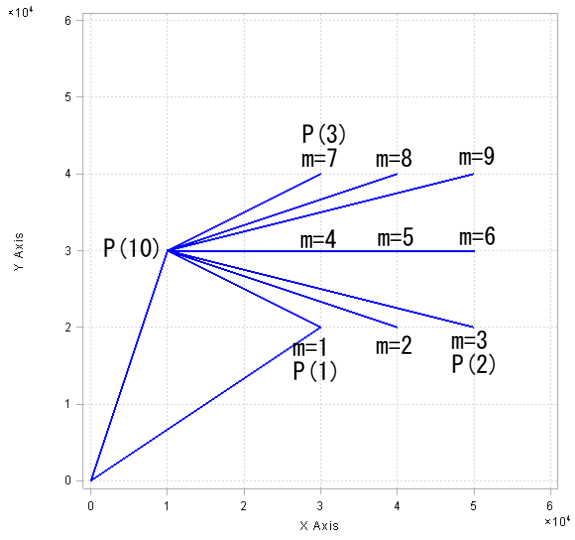
パレタイズ

パレットコーナーの３点または４点からパレット上のマトリックスを演算します。下記はプログラム中で各点を作成していますが、ティーチングやPC・USBメモリから読み込んだ点データでもOkです。
このプログラムはパレット１の各マトリックスとP(10)間を往復します。下図は実行時の軌跡です。

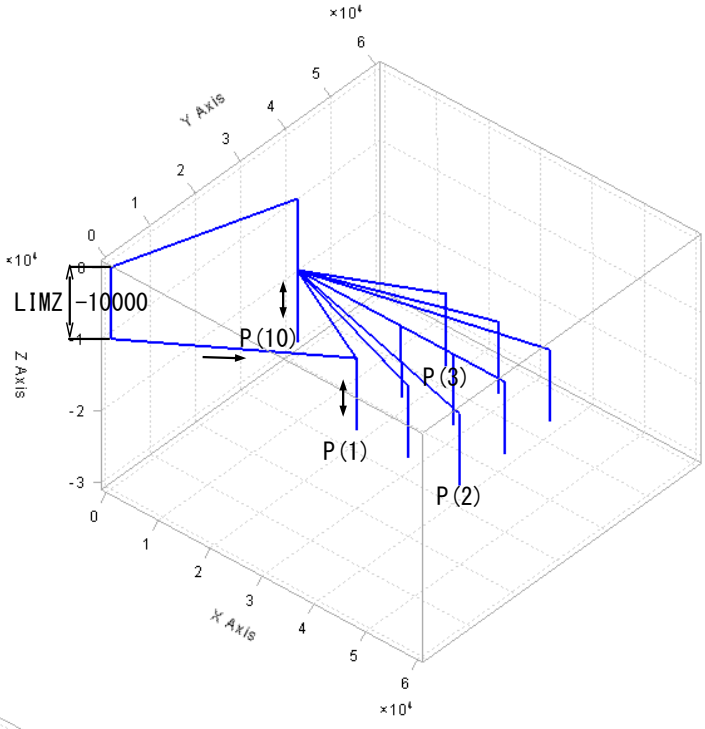
```
SETP 1 30000 20000 0 -20000      /* P(1)
SETP 2 50000 20000 0 -20000      /* P(2)
SETP 3 30000 40000 0 -20000      /* P(3)
SETP 10 10000 30000 0 -20000     /* P(10)
PALLET 1 P(1) P(2) P(3) 3 3      /* パレット宣言 3行3列
LIMZ -10000                      /* JUMP Z上限規制
FOR m=1 TO 9
  JUMP PL(1;m)                   /* パレット1のmに移動
  JUMP P(10)
NEXT
MOVS Z_A 0                       /* Z軸上昇
WAIT RR(ALL_A)==0
MOVL 0 0                         /* XY軸移動
WAIT RR(ALL_A)==0
```



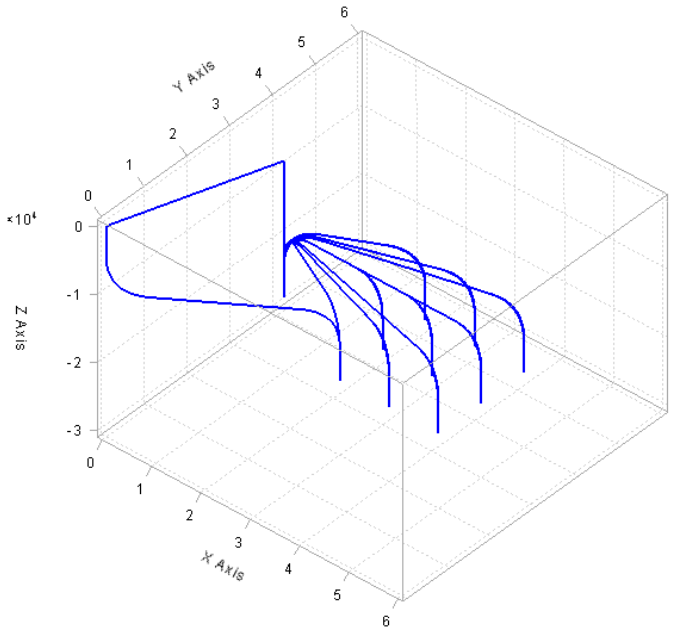
※4点ティーチングすれば歪んだパレットにも対応できます。



XY View



XYZ View



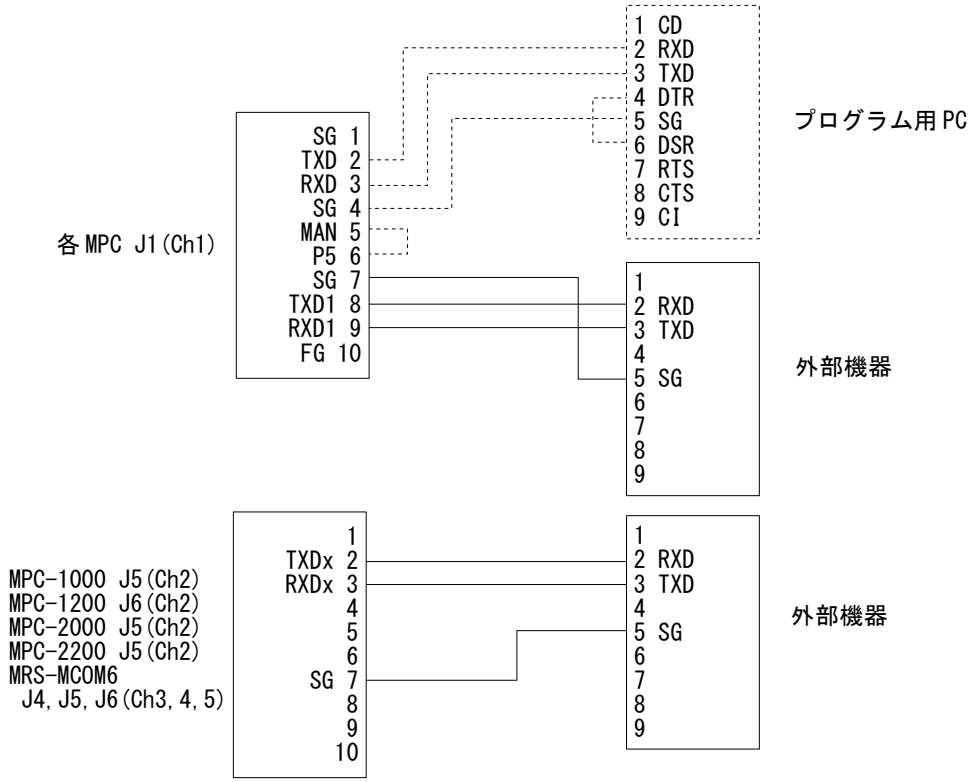
JUMPをWARPにすると角の取れたゲートモーションとなり高速化できます。

```
FOR m=1 TO 9
  WARP 5000 PL(1;m) 5000
  WARP 5000 P(10) 5000
NEXT
```

シリアル通信

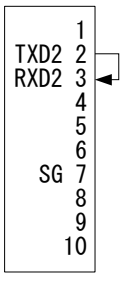
接続例

各 MPC の J1 コネクタにはプログラムポートとユーザーポート Ch1、J5(J6)にはユーザーポート Ch2 が割り当てられています。汎用通信は CNFG#で設定し PRINT#と INPUT#で送受信します。
タッチパネルは MEWNET コマンドです。同一チャンネルで CNFG#と MEWNET の重複はできません。



下は右図のように TXD(送信)と RXD(受信)を接続してループバックしています。これは最も簡単な送受信の確認方法です。ここでは受信文字列中から VAL 関数で数値を抽出しています。
入力バッファクリアに CNFG#を用いるのは禁止です。CNFG#はプログラム冒頭に 1 回だけ実行し、入力バッファのクリアは CLR_BUF で行います。

```
CNFG# 2 "9600b8pns1NONE"                /* 通信設定 最初に1回
DO
  /* CNFG# 2 "9600b8pns1NONE"           /* CNFG#の繰り返し実行はNG
  INPUT# 2 CLR_BUF                      /* 受信バッファクリア
  PRINT# 2 "ABC123.456¥r"               /* 送信
  INPUT# 2 RCV$                         /* 受信
  A=VAL(RCV$)                           /* 文字列内の最初の数値
  B=VAL(0)                              /* 次の数値
  PRINT "RCV$=" RCV$ "A=" A "B=" B      /* 結果表示
  TIME 500
LOOP
```

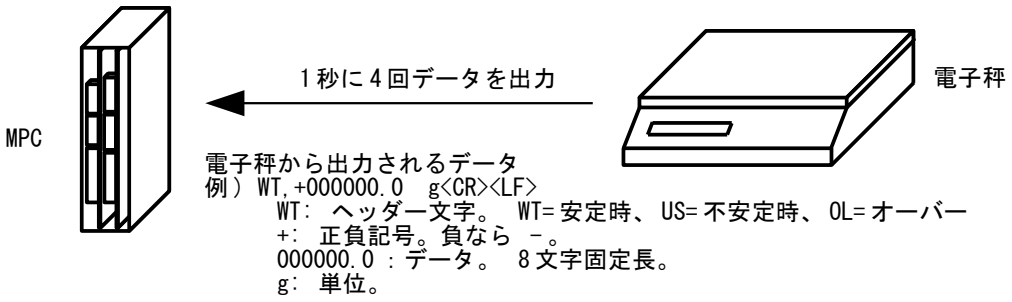


実行結果

```
RUN

# RCV$= ABC123.456 A= 123 B= 456
 RCV$= ABC123.456 A= 123 B= 456
 RCV$= ABC123.456 A= 123 B= 456
```

電子秤の通信例

電子秤から定期的に送られてくるデータから文字と数値を取り出します。INPUT#のデフォルトのデリミタはCR(&HD=13)ですが EOL オプションで LF(&HA=10)にしています。TMOUT オプションでタイムアウトを２秒とし、エラーになったらリトライします。




```
  CNFG# 1 "2400b7pes1NONE"       /* 初期化
  FORMAT ""
  DO
*RETRY
    INPUT# 1 EOL|10 TMOUT|2 RCV$ /* LFまで受信 TMOUT 2秒
    IF rse_<>0 THEN              /* TMOUT処理。rse_は予約変数。
      PRINT "tmout retry"
      GOTO *RETRY
    END_IF
    'PR RCV$
    ptr_=RCV$                    /* 文字列RCV$のポインタ。ptr_は予約変数。
    HEADER$=PTR$(2)              /* RCV$の先頭から2文字をHEADER$へコピー
    ptr_=RCV$+14                 /* ポインタを14文字進める
    UNIT$=PTR$(1)                /* ポインタ位置から1文字をUNIT$へコピー
    SELECT_CASE HEADER$          /* ヘッダーをチェックする
      CASE "WT" : RESULT$="○"
      CASE "US" : RESULT$="△"
      CASE "OL" : RESULT$="×"
      CASE_ELSE                  /* 想定外
        PRINT "invalid header"
        GOTO *RETRY
    END_SELECT
    WEIGHT1$=STR$(VAL(RCV$))     /* 文字列RCV$の中の最初の数値(この場合整数部)
    WEIGHT2$="."+STR$(VAL(0))    /* 文字列RCV$の中の次の数値(この場合小数部)
    PRINT RESULT$ WEIGHT1$ WEIGHT2$ UNIT$  /* FTM表示
  LOOP
```

実行結果

```
RUN

○ 0 .0 g
○ 0 .0 g
△ 51 .3 g   ◀―― 物を乗せる
△ 111 .9 g
△ 117 .0 g
△ 117 .3 g
○ 117 .3 g
○ 117 .3 g
```

STX～ETX 文字列の例

よくある STX(&H02)で始まり ETX(&H03)で終わるの文字列の受信と、その中の数値の和を求めます。
(文字長、数値の位置は固定という想定です。)

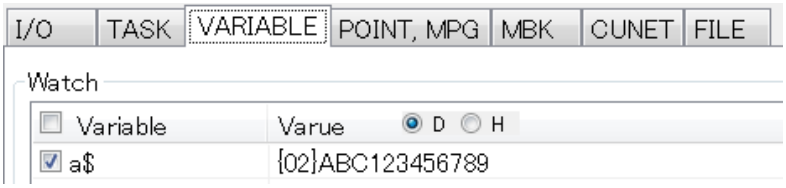
```
CNFG# 2 "38400b8pns1NONE"
DO
  INPUT# 2 CLR_BUF
  INPUT# 2 EOL|3 a$              /* &H03(EXT)まで受信(&H03は含まない)
  ptr_=a$+1                      /* a$の先頭位置+1をポインタにセット
  b$=PTR$(3)                     /* ポインタ位置から3文字取得
  ptr_=ptr_+3                    /* ポインタを+3
  c$=PTR$(9)                     /* ポインタ位置から9文字取得
  sum=0
  FOR i=0 TO LEN(c$)-1           /* c$を1文字ずつ数値に変換して足す
    ptr_=c$+i                      /* c$の中の位置
    sum=sum+VAL(PTR$(1))           /* 1文字を数値に変換して足す
  NEXT
  PRINT "a$=" a$ "b$=" b$ "c$=" c$ "sum=" sum
LOOP
```

```
RUN

# a$= ABC123456789 b$= ABC c$= 123456789 sum= 45
```

実行結果

接続機器から
<STX>ABC123456789<ETX>
を送信します。

↑ ここに<STX>が有りますが見えません。
↓ MPC Monitor では表示します。



バイナリの送受信

送信

'¥x'の後に2桁でバイナリコードを指定します。（サポート 1.14_38 2015/07/28 以降）

```
CNFG# 2 "9600b8pns1NONE"
snd$="¥x02"+"¥x22"+"¥x00"+"¥x0A"+"¥x0D"+"¥x09"+"¥x03"
PRINT# 2 snd$                                          /* (1)
snd$="¥x02¥x22123abc¥x03"
PRINT# 2 snd$                                          /* (2)
```

実行結果

(1) &H02&H22&H00&H0A&H0D&H09&H03
と送信します。

(2) &H02&H22&H31&H32&H33&H61&H62&H63&H03 == &H02&H22'1''2''3''a''b''c'&H03
と送信します。

受信

この例は10キャラクタ固定長のデータ受信です。受信内容の確認やチェックサムなどの後処理がしやすいように1キャラクタづつ点データ配列に格納しています。

```
CNFG# 2 "9600b8pns1NONE"
DO
  INPUT# 2 CHR_C|10 rcv$          /* 10キャラ受信
  ptr_=rcv$                       /* ポインターにrcv$の文字位置をセット
  FOR p=100 TO 109
    X(p)=ASC(1)                   /* ポインター位置のキャラクタコード
    ptr_=ptr_+1                   /* ポインターインクリメント
  NEXT
  FOR p=100 TO 109                /* 確認表示
    PR "X(" p ")=" HEX$(X(p))
  NEXT
LOOP
```

```
RUN
                          接続機器から
                          &H00&H01&H02&H03&H04&H11&H22&HAA&HCC&HFF
                          が送られてくる
# X( 100 )= 00000000
 X( 101 )= 00000001
 X( 102 )= 00000002
 X( 103 )= 00000003
 X( 104 )= 00000004
 X( 105 )= 00000011
 X( 106 )= 00000022
 X( 107 )= 000000AA
 X( 108 )= 000000CC
 X( 109 )= 000000FF
```

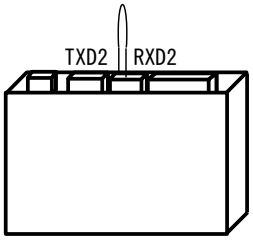
実行結果

通信内容の確認

RS コマンドで送受信バッファをダンプします。

テストプログラム

```
QUIT_FORK 1 *main
END
*main
CNFG# 38400 2
s$=CHR$(2)+"ABC123"+CHR$(3)      /* 送信文字列
INPUT# 2 CLR_BUF                 /* 受信バッファクリア
PRINT# 2 s$                      /* 送信
INPUT# 2 EOL|3 r$                /* 受信
PRINT r$
```



TXD2（送信）と RXD2（受信）をショート（ループバック）してプログラムを実行しました。

実行結果

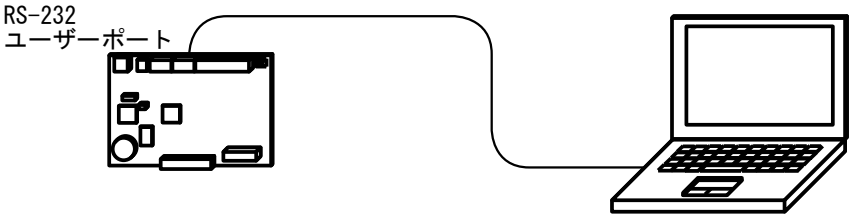
```
RUN

# ABC123 ←PRINT r$ の表示
RS 2     ←RS コマンドをダイレクト実行

CH ==>2
The input buffer is empty!
Dump input buffer
0       <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
20      <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
40      <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
60      <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
80      <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
100     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
120     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
140     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
160     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
180     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
200     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
220     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
240     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><STX>  A    B    C    1    2    3  <ETX> <---Most recent
                                                  受信したキャラクタ
Dump output buffer
0       <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
20      <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
40      <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
60      <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
80      <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
100     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
120     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
140     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
160     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
180     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
200     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
220     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL> <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL>
240     <NUL><NUL><NUL><NUL><NUL><NUL><NUL><NUL><STX>  A    B    C    1    2    3  <ETX> <---Most recent
                                                  送信したキャラクタ
```

ツールを使った RS-232 通信の確認

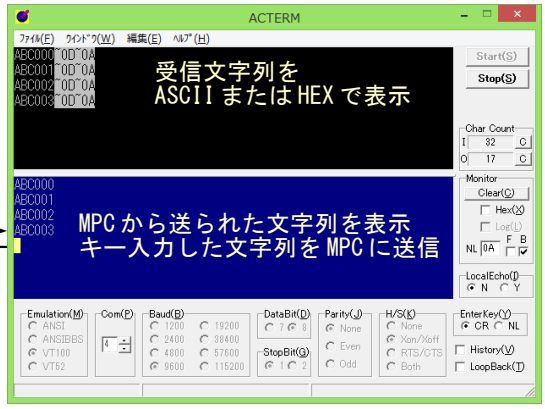
パソコンのアプリケーションを使って MPC の送受信をパソコンで確認できます。これらのアプリはインストールで FTMW2K などと同じフォルダにセットアップされます。

汎用ターミナルソフト ACTERM

接続機器の代わりにパソコンを使って MPC の送受信状態を確認できます。

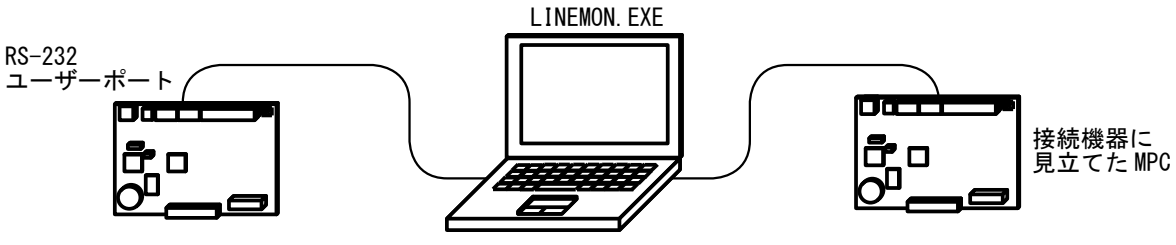



```
CNFG# 1 "9600b8pns1NONE"      /* 通信設定
i=0
DO
  FORMAT "000"
  SEND$="ABC"+STR$(i)+"¥r¥n"
  INPUT# 1 CLR_BUF            /* 受信バッファクリア
  PRINT# 1 SEND$              /* 送信
  INPUT# 1 RCV$               /* 受信
  PRINT RCV$                  /* 結果表示
  i=i+1
LOOP
```

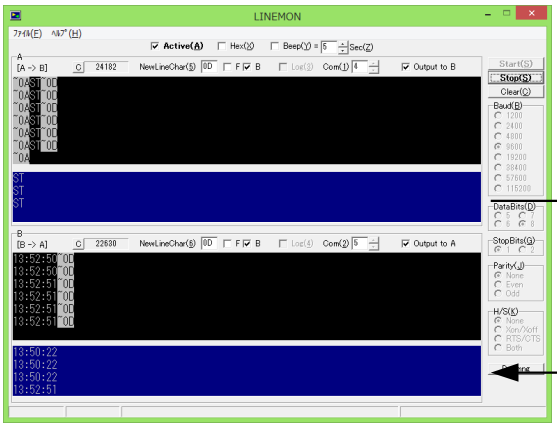


ラインモニター LINEMON

MPC と接続機器の間にパソコンを入れて双方向の通信内容をモニターします。
RS-232 ポートが 2 つ必要です。




```
CNFG# 1 "9600b8pns1NONE"
DO
  SEND$="ST¥r¥n"
  INPUT# 1 CLR_BUF
  PRINT# 1 SEND$
  INPUT# 1 RCV$
  PRINT RCV$
LOOP
```



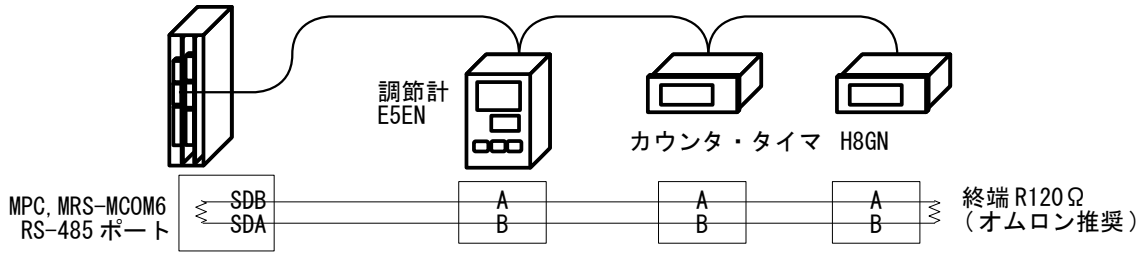

```
CNFG# 2 "9600b8pns1NONE"
DO
  INPUT# 2 a$
  PRINT# 2 TIME$(1) "¥r"
LOOP
```

RS-485

RS-485 はフェイルセーフ回路を内蔵しています。CNFG#で RS485 を指定すればトーカー・リスナーを自動制御します。コマンドは RS-232 と共通です。
RS-485 通信機器とマルチドロップ接続が可能です。

例 1) オムロン温調器と電子カウンタ

オムロンデジタル調節計と電子カウンタ・タイマのマルチドロップ接続例です。
E5EN の変数エリアの現在値(温度)を読み込みます。CompoWay/F プロトコルのフォーマットに従って文字列を組立て、BCC を計算して送信、受信データから BCC を計算し必要な部分を切り出します。




```
CNFG# 5 RS485 "9600b7pes2NONE"  /* MRS-MCOM CH5 RS485 設定
FORMAT ""                       /* 文字列フォーマット無し
SEND$=CHR$(2)                   /* STX
SEND$=SEND$+"01"                /* ノード番号
SEND$=SEND$+"000"               /* サブアドレス,SID
SEND$=SEND$+"0101"              /* MRC,SRC
SEND$=SEND$+"C0"                /* 変数種別
SEND$=SEND$+"0000"              /* 開始アドレス
SEND$=SEND$+"00"                /* ビット位置
SEND$=SEND$+"0001"              /* 要素数
SEND$=SEND$+CHR$(3)             /* ETX
PUT_BCC=0                       /* 送信BCCを計算
FOR I=1 TO LEN(SEND$)-1
  STRCPY SEND$ BUF$ I 1
  PUT_BCC=PUT_BCC^ASC(BUF$)&&HFF  /* 排他的論理和
NEXT I
PRINT# 5 SEND$ CHR$(PUT_BCC)    /* 送信
DO
  INPUT# 5 CHR_C|1 BUF$         /* 1文字ずつ受信
  IF ASC(BUF$)==&H02 THEN       /* STX(データの先頭)待ち
    BREAK
  END_IF
LOOP
GET_STR$=""                     /* 受信文字変数(レスポンスフレームSTX以降ETXまで)
DO
  INPUT# 5 CHR_C|1 BUF$         /* 1文字ずつ受信
  GET_STR$=GET_STR$+BUF$
  IF ASC(BUF$)==&H03 THEN       /* ETX受信ならLOOP抜ける
    BREAK
  END_IF
LOOP
INPUT# 5 CHR_C|1 GET_BCC0$      /* 1文字(BCCデータ)受信
GET_BCC0=ASC(GET_BCC0$)         /* 受信したBCCデータ->数値
GET_BCC1=0
FOR I=0 TO LEN(GET_STR$)-1      /* 受信文字列からBCCを計算
  STRCPY GET_STR$ BUF$ I 1
  GET_BCC1=GET_BCC1^ASC(BUF$)&&HFF
NEXT I
IF GET_BCC0<>GET_BCC1 THEN
  PRINT "BCC ERROR"
  PRINT "受信BCC=" HEX$(GET_BCC0) " 計算BCC=" HEX$(GET_BCC1)
  END
END_IF
STRCPY GET_STR$ NODE$ 0 2       /* 0から2文字がノードNo
STRCPY GET_STR$ GET_TMP$ 14 8   /* 14から8文字が温度
```

CompoWay/F 通信マクロコマンドを用いると、文字列の組立てが簡単に、BCC の計算が不要になります。

送信手順
1) COMPOWAY コマンドで送信するテキストを構築します。
2) PRINT# コマンドに COMPOWAY オプションを与えて実行すると STX と ETX、BCC を付加したコマンドフレームを送信します。

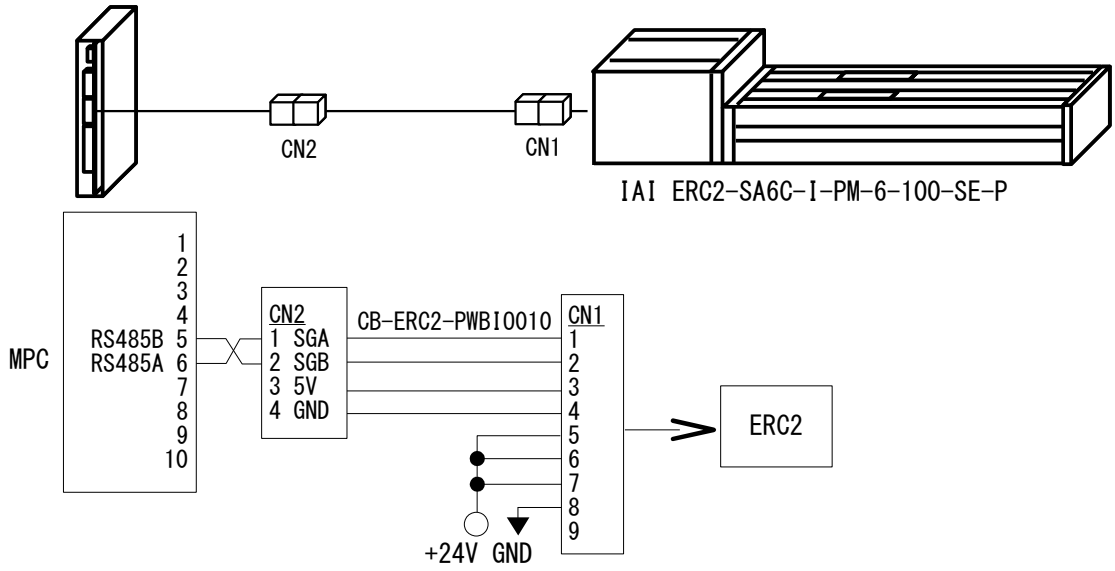
受信手順
1) INPUT#コマンドに COMPOWAY オプションを与えて実行するとレスポンスフレームを受信し、BCC を計算します。
2) COMPOWAY コマンドでレスポンスフレームから要素を変数に展開します。

下記は COMPOWAY マクロコマンドでの通信例。文字列処理にはポインタを使っています。

```
CNFG# 5 RS485 "9600b7pes2NONE"      /* 通信初期化
FORMAT ""                           /* 文字列フォーマット無し

/* コマンドフレームのテキスト部分の要素を変数・文字列変数に入れています。
node_no=1                           /* ノード no
sub_adr=0                           /* サブアドレス
sid=0                               /* SID

mrc_src$="0101"                     /* MRC,SRC
hensu_shu$="C0"                     /* 変数種別
str_adr$="0000"                     /* 開始アドレス
bit_ichi$="00"                      /* bit位置
yoso_su$="0001"                     /* 要素数
setteichi$=""                       /* 設定値 無し

/* コマンドテキストを作成します
cmnd_txt$=mrc_src$+hensu_shu$+str_adr$+bit_ichi$+yoso_su$+setteichi$
/* ノードNoからコマンドテキストまで結合して snd$ に入れます
COMPOWAY node_no sub_adr sid cmnd_txt$ snd$
/* コマンドフレームを送信します
PRINT# 5 COMPOWAY snd$

/* レスポンスフレームをrcv$に受信します
INPUT# 5 COMPOWAY TMOUT|2 rcv$
/* res$にコマンドテキストの文字列が入ります
COMPOWAY rcv$ node_no sub_adr end_code res$

/* res$ の0から数えて4文字目から4文字がレスポンスコードです
ptr_=res$+4                         /* ptr_ はポインタ予約変数。res$の4文字目を指す
res_code=HEX(PTR$(4))               /* ptr_の位置から4文字コピー

/* res$ の0から数えて8文字目から8文字が読出データです
ptr_=res$+8                         /* ポインタはres$の8文字目を指す
res_data$=PTR$(8)                   /* ptr_の位置から8文字コピー
PRINT res_code HEX(res_data$)       /* 温度表示
```

例 2) IAI ロボシリンダ

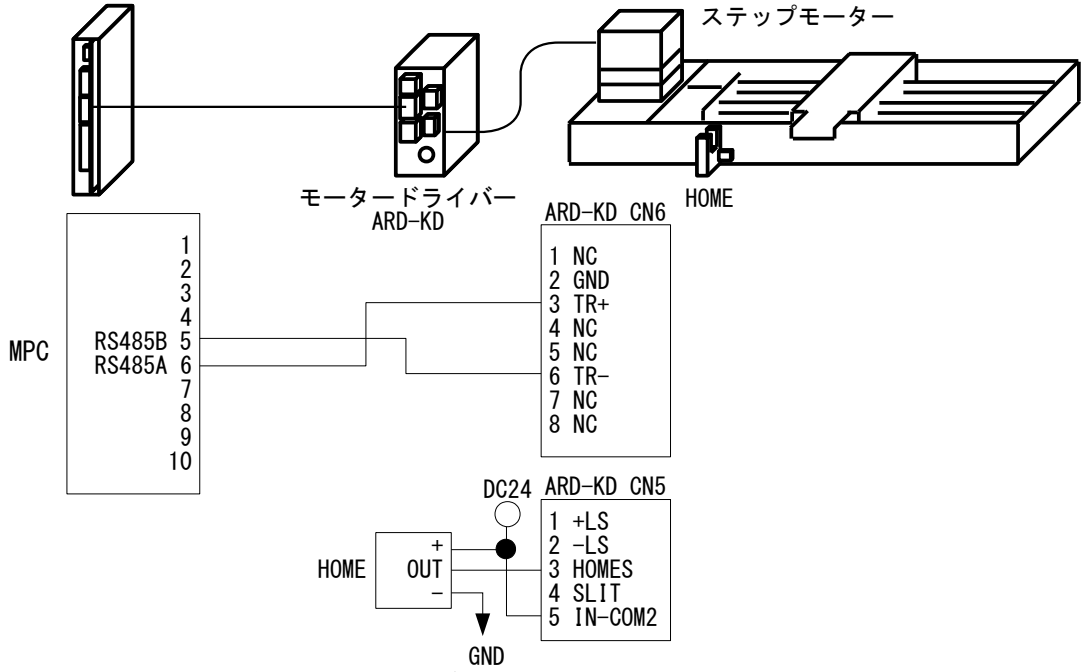
RS-485 Modbus ASCII または Modbus RTU で制御します。



事例はホームページのアプリケーションノートをご参照ください。
http://departonline.jp/mpc2000/ref/headline/appendix/pdf/an2k-029.pdf

例 3) オリエンタルモーター ARD-KD

RS-485 Modbus RTU モードで制御します。



事例はホームページの技術情報をご参照ください。
http://departonline.jp/mpc2000/ref/headline/appendix/pdf/mpc-2100_rs485_stepmotor.pdf

＊＊＊

MPC がマスター、装置がスレーブで、RS-485 マルチドロップ接続により多軸制御が可能です。(ARD-KD は取り説に”上位システムに対して最大 31 台まで接続できます”とあります。)

マスターによるスレーブレジスタの読み書き(クエリー)で動作します。レジスタ、クエリーはメーカーにより異なります。

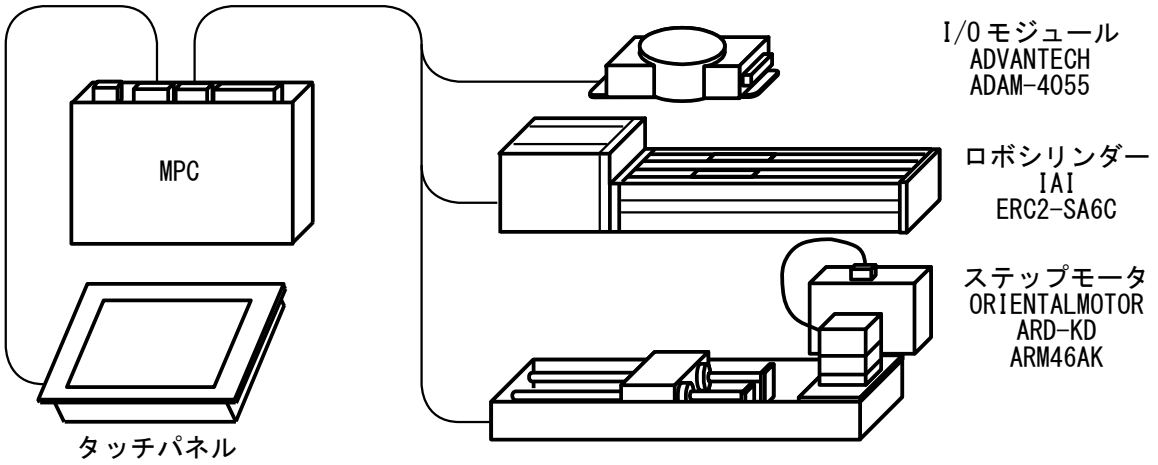
通信チェックは CRC-16 で行います。

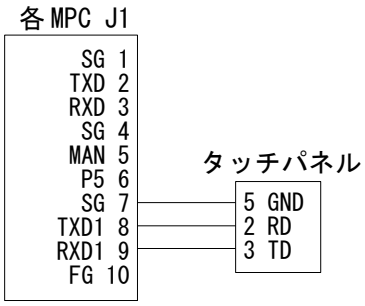
Modbus とは Modicon 社によって開発されたオープン Master/Slave アプリケーションプロトコルです。
RTU は Remote Terminal Unit(遠隔端末装置) 、CRC-16 は Cyclic Redundancy Check 16 (巡回冗長検査)です。

例 4)Modbus-RTU クエリ送受信

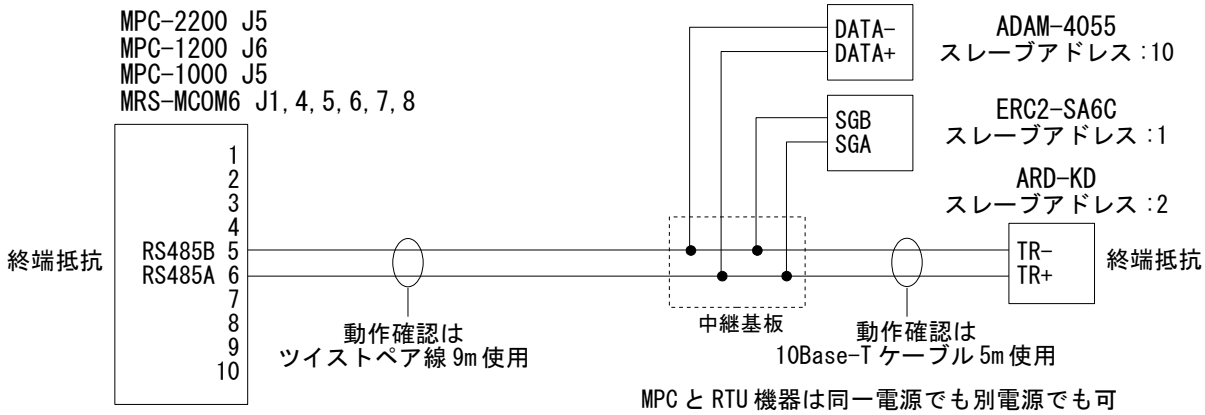
　専用コマンドと関数で容易にクエリの送受信を行えます。
X_RTU にスレーブアドレス・ファンクションコード・データを設定して送信、R_RTU(n)で受信後 R_RTU コマンドまたは R_RTU(s,i)でデータを取り出します。CRC-16 はコマンド・関数で自動的に計算されます。

下は異なるメーカーの異なるクエリフォーマットの機器を 1 本の通信ラインで動かした例です。





RTU 機器のアドレス、通信は専用ツールや DSW で設定
通信仕様は全部同じ
38400bps, 8bit, PrityNone, 1Stop, XON/XOFFNone



MPC と RTU 機器は同一電源でも別電源でも可

[参考資料] 技術情報 「Modbus-RTU 制御」

USB メモリ

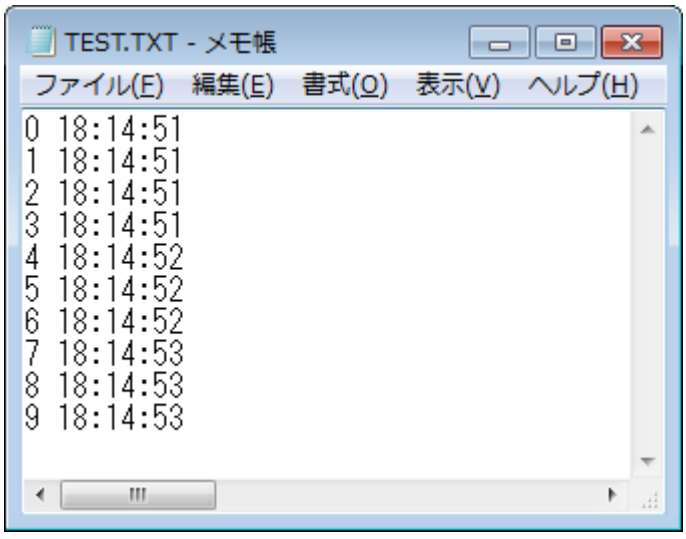
記録や機種切替時のデータ変更などに利用できます。

ファイルの読み書き

```
FILE$="TEST.TXT"                    /* ファイル名
USB_DEL FILE$                       /* Delete
FOR i=0 TO 9
  st$=STR$(i)+" "+TIME$(1)+"¥n"     /* 文字列生成
  USB_WRITE  st$                    /* USBメモリに書き込み
  TIME 100
NEXT
WHILE EOF(0)<>1
  USB_READ r$                       /* USBメモリから読込
  PR r$
WEND
OFF_USB                             /* Close
PRINT "Owari"
```

実行結果

```
RUN

# 0 18:14:51
 1 18:14:51
 2 18:14:51
 3 18:14:51
 4 18:14:52
 5 18:14:52
 6 18:14:52
 7 18:14:53
 8 18:14:53
 9 18:14:53

 A:>
 Owari
```



USB メモリを PC に差してメモ帳で開く

点データの保存･読込

点データ、MBK データの保存・読み込みは、パソコンとのデータ受け渡しに便利です。

保存

```
FILE$="TEST.P2K"                    /* ファイル名
USB_DEL FILE$                       /* USBメモリーの削除
FOR i=1 TO 10                       /* 点データ作成
  SETP i i i i i
NEXT
USB_PSAVE P(1) 10 FILE$             /* USBメモリーに書き込み
```

```
RUN

#DIR
2015/08/31  10:52                   175 TEST.P2K
          2 個のファイル                175 バイト
          0 個のディレクトリ
A:>
```

実行後 DIR コマンドでファイルを確認

```
 SETP 1 1 1 1 1↓
 SETP 2 2 2 2 2↓
 SETP 3 3 3 3 3↓
 SETP 4 4 4 4 4↓
 SETP 5 5 5 5 5↓
 SETP 6 6 6 6 6↓
 SETP 7 7 7 7 7↓
 SETP 8 8 8 8 8↓
 SETP 9 9 9 9 9↓
 SETP 10 10 10 10 10↓
[EOF]
```

メモリーをパソコンに差し替えてテキストエディターで TEST.P2K を開く

読込

```
FILL P(1) 10                        /* 点データ使用領域初期化
GOSUB *PNT_PRINT                    /* 点データ表示
USB_PLOAD "TEST.P2K"                /* USBメモリーから読込
GOSUB *PNT_PRINT
END
*PNT_PRINT
FOR i=1 TO 10
  PRINT "P(" i ")=" P(i)
NEXT
RETURN
```

```
RUN

# P( 1 )= 0 0 0 0
 P( 2 )= 0 0 0 0
 P( 3 )= 0 0 0 0
   （中略）
 P( 1 )= 1 1 1 1
 P( 2 )= 2 2 2 2
 P( 3 )= 3 3 3 3
   （後略）
```

実行結果

LOAD 前の点データ

LOAD 後の点データ

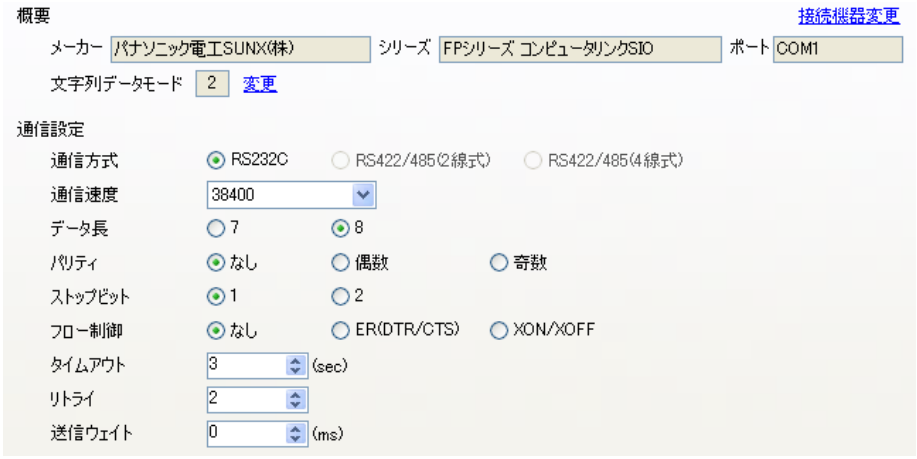
タッチパネル

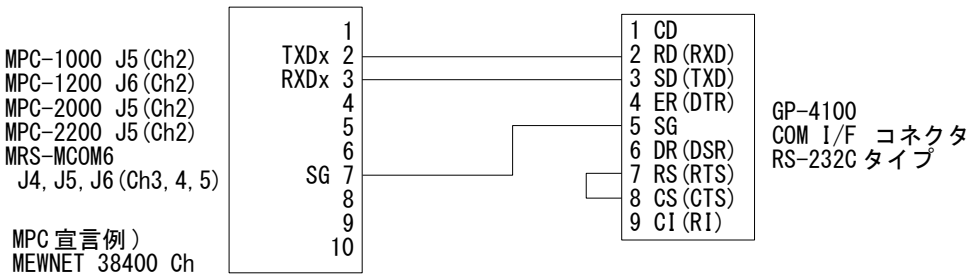
タッチパネルもシリアル通信機器のひとつですが、MPC はパナソニック MEWNET 準拠のプロトコルを内蔵しており、宣言のみでリンクし I/O と同様のコマンドで制御します。

デザイナーの通信設定と結線例

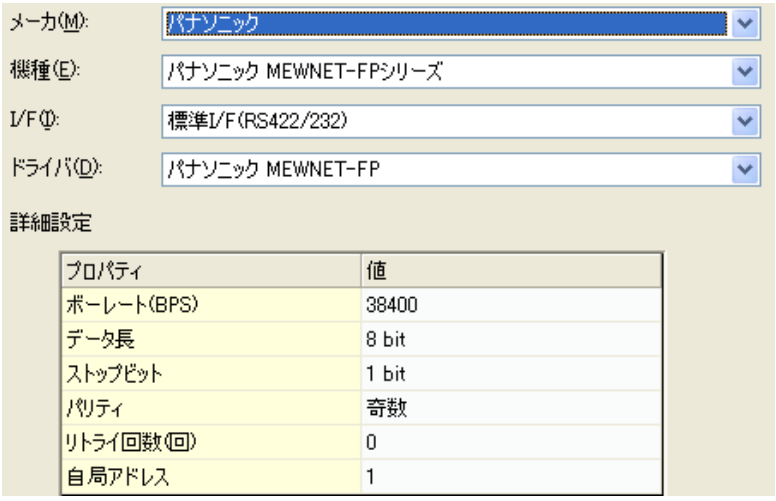
この他にタッチパネル本体の設定等が必要です。

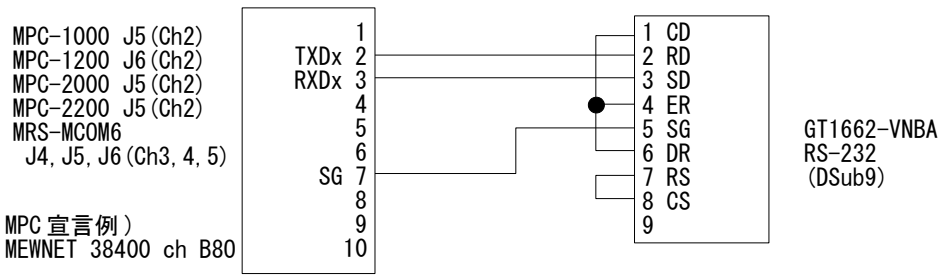
デジタル GP-4000 シリーズ



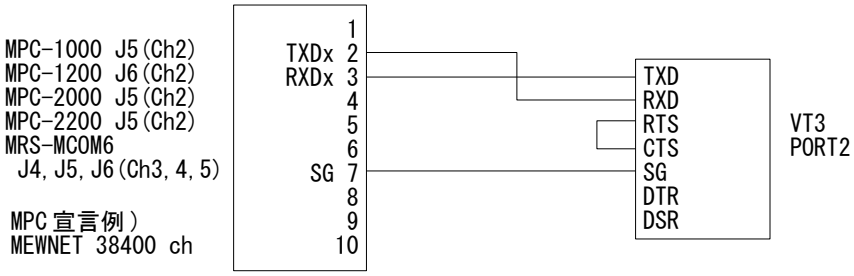


三菱 GOT-1000 シリーズ

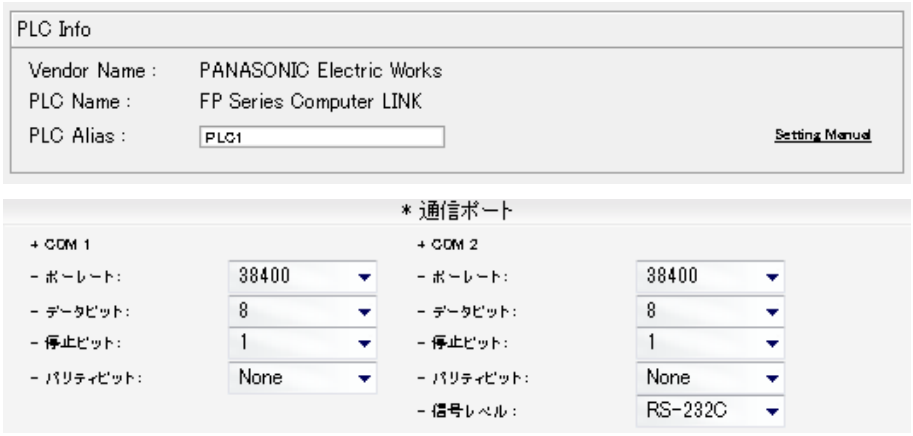


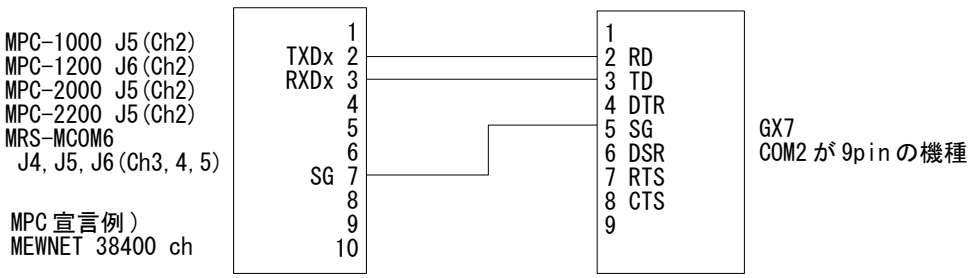


キーエンス VT3 シリーズ



ミスミ GX7 シリーズ

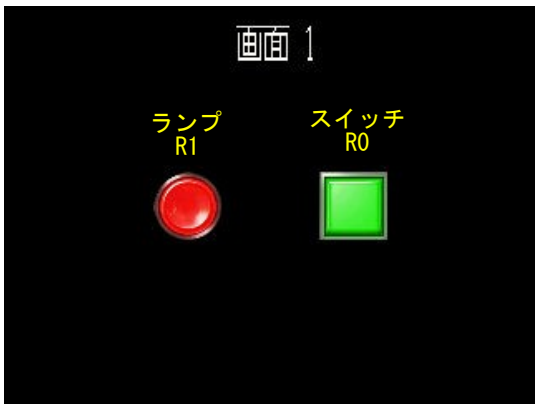




[参考資料]　　技術情報　「タッチパネルの使用例」

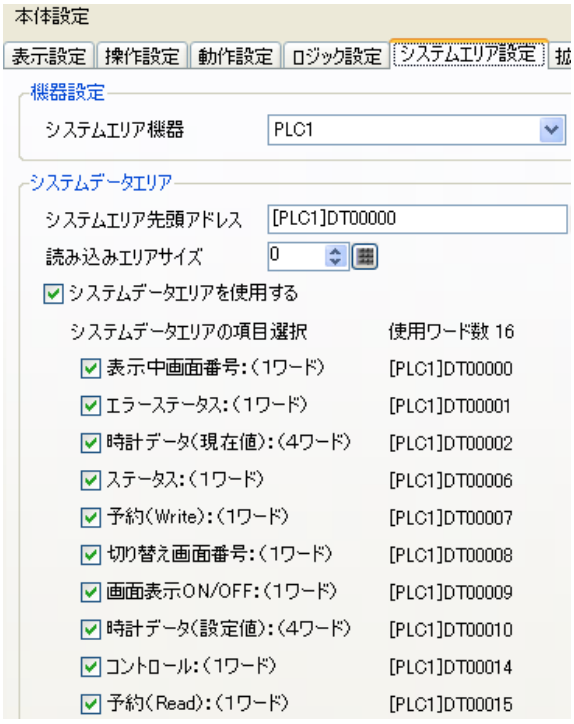
デジタル GP-4301 のデザインとプログラム例

ベース画面 1



ベース画面 2

デジタル GP シリーズは、本体設定の
「システムデータエリアを使用する」
で MPC から画面の切替が可能になります。
下の場合は DT8 が画面切替です。
この設定は各社異なります。



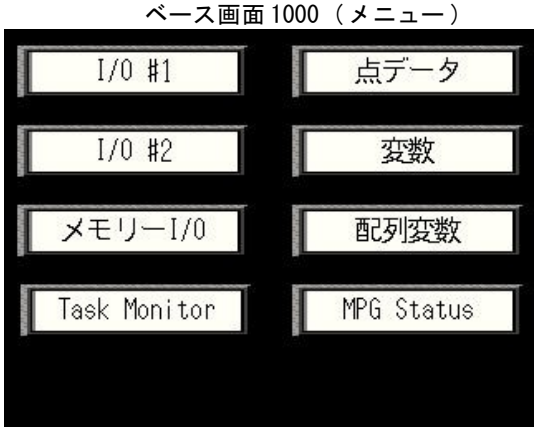
DT エリアは MBK(),S_MBK、R エリアは I/O コマンド (SW(),IN(),ON,OFF,OUT) で操作します。
R エリアは 70000 番台に割り当てられています。

```
MEWNET 38400 2              /* タッチパネル通信設定
TIME 1000
MBK(8)=1                    /* ベース画面 1 表示
QUIT_FORK 2 *LampFlick      /* ランプ点滅タスク起動
DO
  IF SW(70000)==1 THEN      /* スイッチ R0 がオンなら
    MBK(8)=2                /* ベース画面 2 表示
    QUIT_FORK 2 *Clock      /* 時計表示タスク起動
    WAIT SW(70000)==0
  END_IF
  IF SW(70002)==1 THEN      /* スイッチ R2 がオンなら
    MBK(8)=1                /* ベース画面 1 表示
    QUIT_FORK 2 *LampFlick  /* ランプ点滅タスク起動
    WAIT SW(70002)==0
  END_IF
  SWAP
LOOP
*LampFlick                  /* ランプ点滅タスク
DO
  ON 70001                  /* ランプ R1 オン
  TIME 500
  OFF 70001                 /* ランプ R1 オフ
  TIME 500
LOOP
*Clock                      /* 時計表示タスク
DO
  MBK(100~Lng)=SYSCLK       /* DT100から32bitにSYSCLKの値を入れる
  S_MBK TIME$(1) 200 8      /* DT200から8文字に時間文字列を入れる
  SWAP
LOOP
```

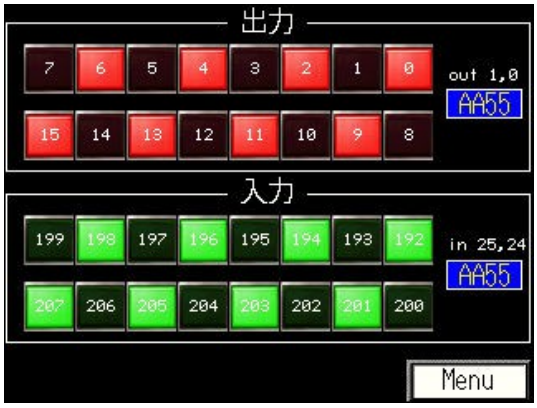
タッチパネルによる稼動状況モニター

MBK(DT) エリア、I/O(R)エリアの上位には、I/O、点データ、MPG の状態など各種情報が割り当てられており、MPC のプログラムを介さずにタッチパネルのデザインだけで参照、変更が可能です。
I/O チェック、稼動状況のモニター、パラメータの変更などに利用できます。
下は GP-4301 のサンプルプロジェクトです。

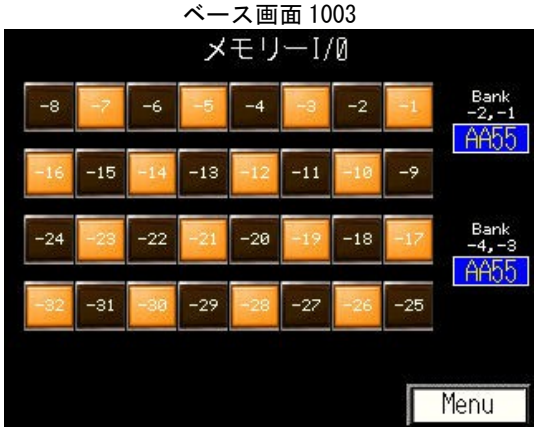
ベース画面 1000（メニュー）



ベース画面 1001



ベース画面 1003



ベース画面 1008

ベース画面 1004

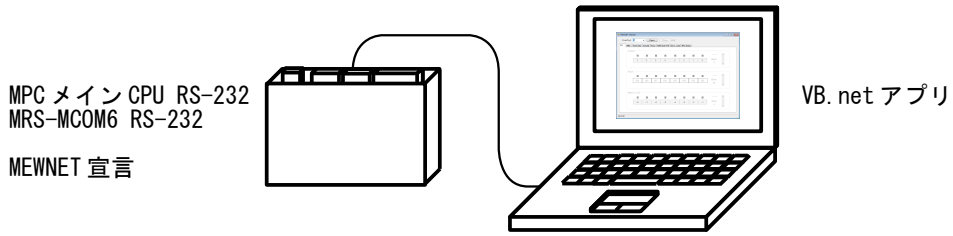
ベース画面 1005

ベース画面 1006

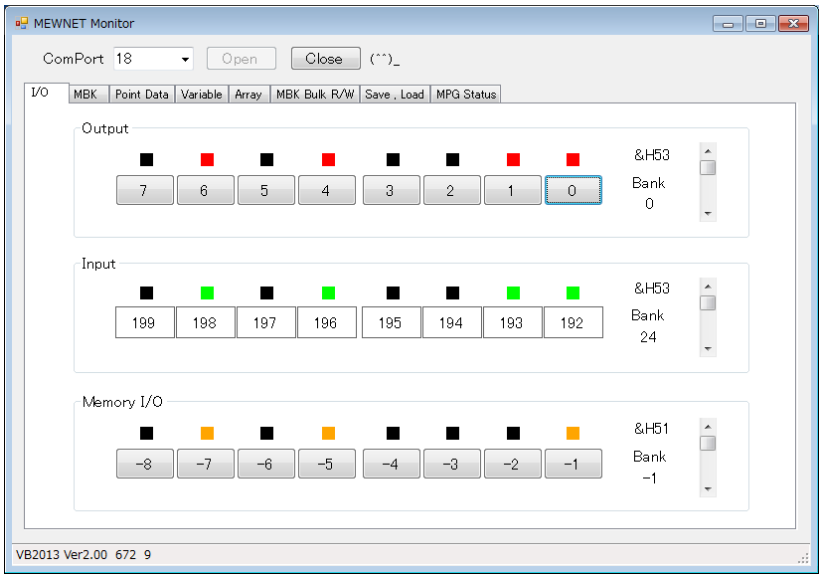
ベース画面 1007

パソコンの MEWNET プロトコル接続

パソコンからも MEWNET プロトコルでリンクできます。I/O チェック、稼動モニターやパートタイムで接続して MPC に記録したデータの取り出しや段取替え時のデータ入れ替えなどに利用できます。
MEWNET プロトコルを利用すれば MPC 側の通信に関するプログラミングは要りません。VB.net 用のクラスモジュールを提供しています。
異なるポートでタッチパネルとの併用も可能です。



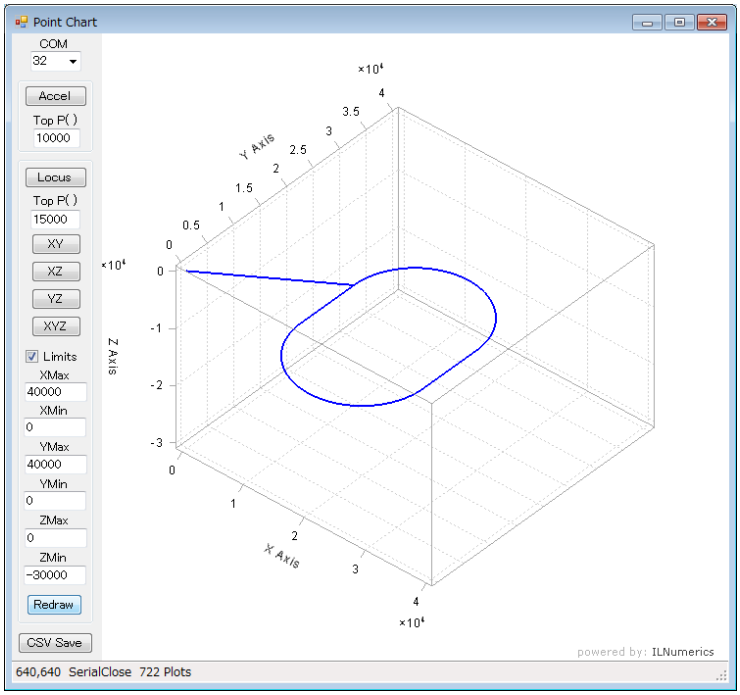
- モニターのサンプル(MEWNET Monitor)



I/O チェック
点データ参照・変更
変数値参照・変更
PG ボードステータス参照
などタッチパネルのモニターと同じことができます。

点データ、MBK データの保存・読込もできます。

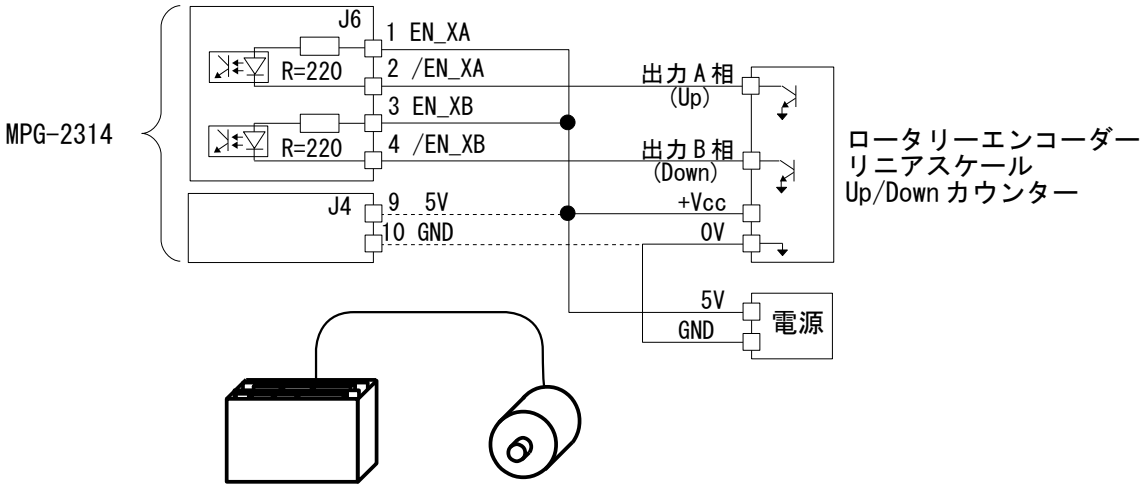
- ILNumerics ライブラリを使った 3D トレーサーのサンプル (加減速・軌跡チャートツール)



MOVT コマンドによる円弧・直線連続補間移動の軌跡を 3D で描きました。
視点を変えて見ることができます。

- これらのソースコードはホームページの「開発ツール ダウンロード」に掲載しています。
  http://www.departonline.jp/acceleng/dev_uty.php

エンコーダー、カウンター入力

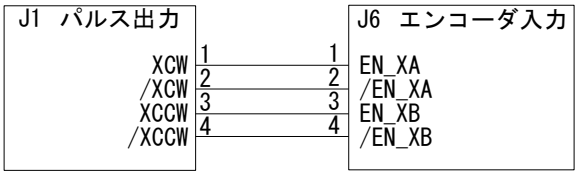


ロータリーエンコーダーの入力例

エンコーダーを回すと 100 カウント毎に出力を On/Off します。

```
PG 0
INSET PHASE1                /* 1逓倍 1000P/Rのエンコーダーなら1回転で1000カウント
/* INSET PHASE2              /* 2逓倍                 〃                2000カウント
/* INSET PHASE4              /* 4逓倍                 〃                4000カウント
CLRPOS -1                   /* カウンタークリア
OUT 0 0
DO
  NOW_XC=X(-1)              /* カウンター値読み込み
  IF (NOW_XC%100)==0 THEN
    OUT @SW(0) 0            /* 出力反転 on/off
    PRINT NOW_XC SW(0)      /* 表示
    WAIT NOW_XC<>X(-1)      /* 変化待ち
  END_IF
LOOP
```

Up/Down カウンター入力例



このサンプルは右図のように MPG-2314 のパスル出力をカウンター入力に接続します。実行するとパルス発生終了毎に X カウンターの値を表示します。
Up/Down カウンターは常に 1 逓倍です。

```
PG 0
ACCEL X_A 10000
INSET UP_DWN                /* Up/Downカウントモード
CLRPOS -1                   /* カウンタークリア
CLRPOS                      /* パルス現在値クリア
P=10000
DO
  P=P*-1
  MOVS X_A P                /* パルス出力
  WAIT RR(X_A)==0           /* パルス出力終了待ち
  PRINT X(-1)               /* カウンター値表示
  TIME 100
LOOP
```

この他に、ロータリーエンコーダーでパルスモーターを回す、移動中にトリガ信号を出力する、リングカウンターとして使う、などの機能があります。

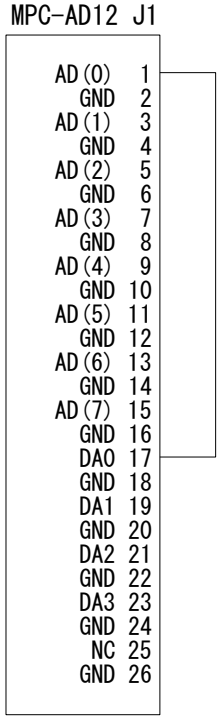
[参考資料]　アプリケーションノート「MPG-2314 のエンコーダ入力と制御例」

AD/DA 変換

MPC-AD12 は 12bit AD 入力が 8 点、12bit DA 出力が 4 点です(標準仕様)。下のサンプルは右図のように DA 出力を AD 入力に接続して実行したものです。

入力方法にはこれら他に、8msec 平均 AD、1msec/2msec 連続 AD、8Ch パラレル AD、トリガ信号同期 AD 等の入力方法があります。

MPC-1200 には 10bit 簡易 AD が 4ch、簡易 DA が 1ch あります。



DA 出力、AD()入力

基本的な AD 入力です

```
*ADDA
  FOR v=0 TO 4000 STEP 500     /* 500mVずつ上昇
    DA v 0                     /* DA Ch0 に出力
    TIME 5
    PRINT AD(0)                /* AD Ch0 の値を表示
  NEXT
```

実行結果

```
RUN

# 0
 499
 1000
 1499
 2001
 2499
 3001
 3499
 4002
```

GET_AD 入力

GET_AD コマンドはサンプリングしたデータを配列に記録します。データの参照や平均化といった後処理に便利です。このサンプルは*DA タスクの出力を*AD タスクで入力し X()に記録します。

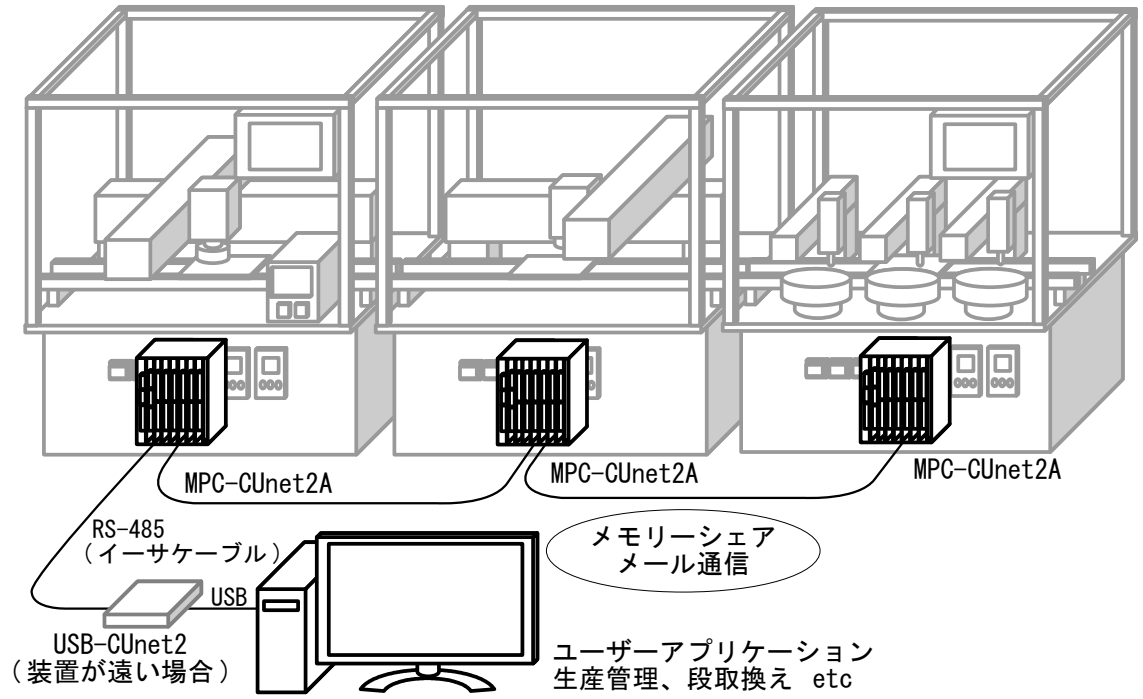
```
*AD
  DA 0 0
  TIME 50
  FILL X(100) 100 0            /* X(100)~X(199) クリア
  QUIT_FORK 2 *DA
  GET_AD 0 X(100) 100 2        /* AD Ch0 入力を X(100)から100点 2msec間隔で記録
  WAIT GET_AD(0)==0            /* 入力終了待ち
  FOR x=100 TO 199
    PRINT "X(" x ")=" X(x)
  NEXT
  END
*DA                            /* DA出力タスク 1msecで20mV上昇
  FOR v=0 TO 4000 STEP 20
    oldsysclk=SYSCLK
    DA v 0
    WAIT oldsysclk<>SYSCLK     /* 1msec待ち
  NEXT
  END
```

実行結果

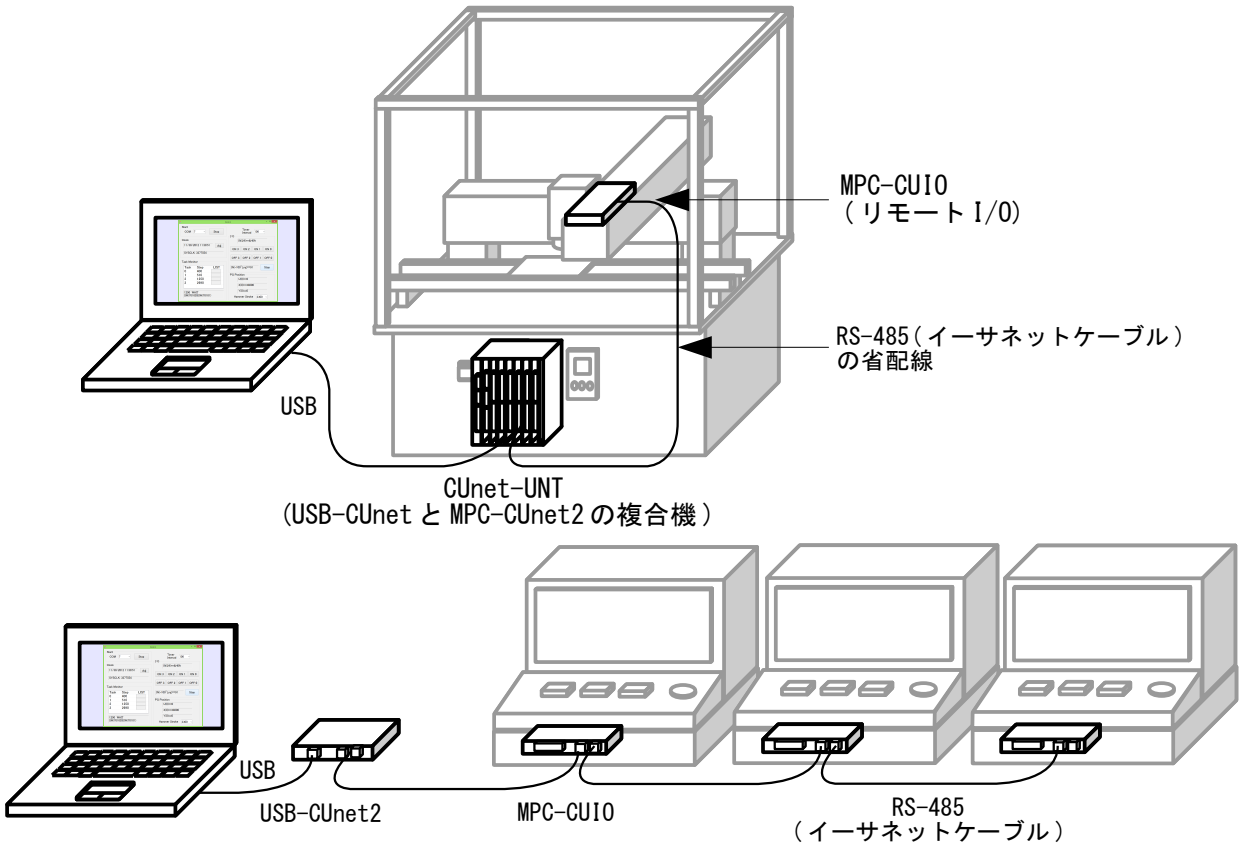
```
RUN

# X( 100 )= 0
 X( 101 )= 41
 X( 102 )= 81
 X( 103 )= 120
 X( 104 )= 161
 X( 105 )= 201
 X( 106 )= 240
```

CUnet

CUnet はステップテクニカ社のオリジナル通信方式で、シリアル通信回線で接続された複数の拠点(ステーション)間で、同一メモリ空間をリアルタイムに共有するリモートメモリ共有技術です。
512byte のメモリーを I/O と同等に扱えます。メール通信により各ステーション間で点データ、MBK データの授受が行えます。　(※Unet 経由でのプログラム転送・デバッグはできません)



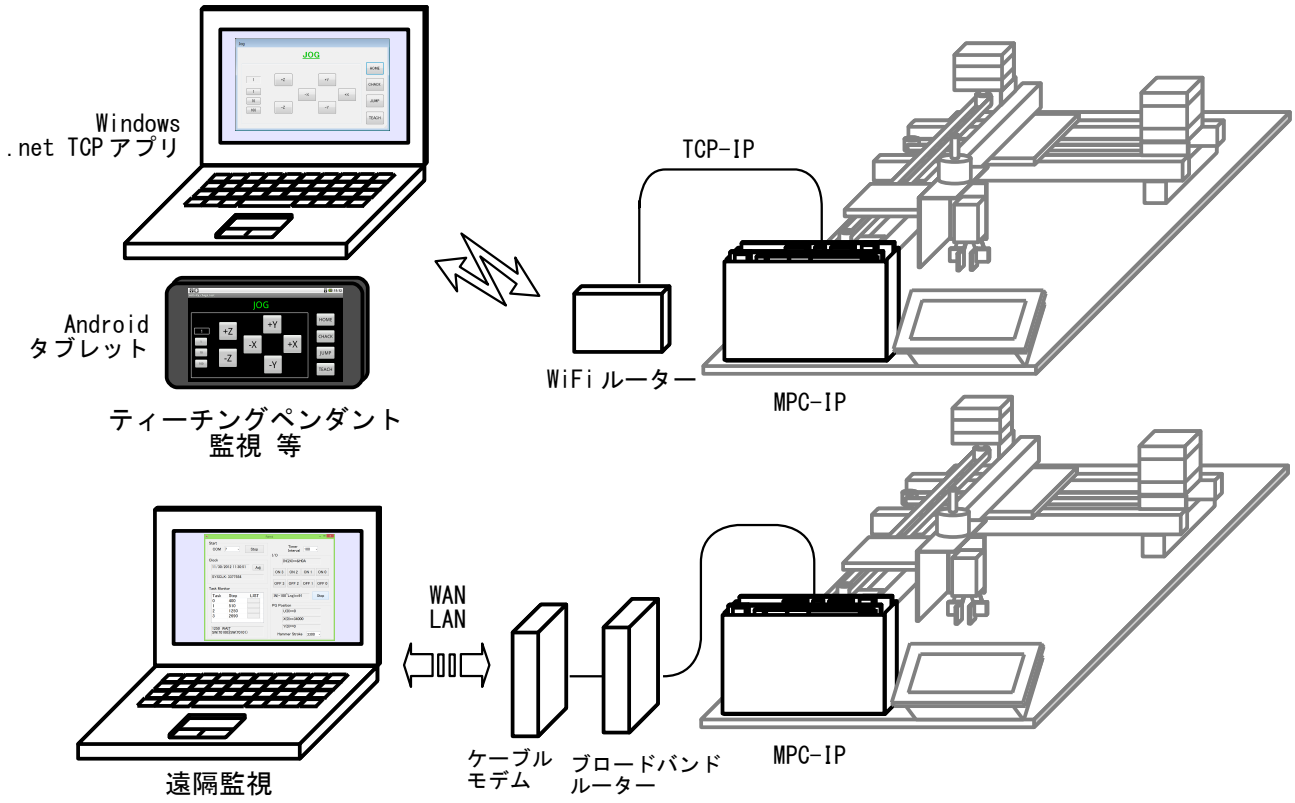
MPC-CUIO は入力 16 出力 16 点を備えたリーモート I/O モジュールです。MPC の遠隔・拡張 I/O です。パソコンのアプリケーションで制御することもできます。



[参考資料]　製品別マニュアル「CUnet 機器 製品別マニュアル」
アプリケーションノート「MPC-CUIO サンプルアプリケーション」

MPC-IP

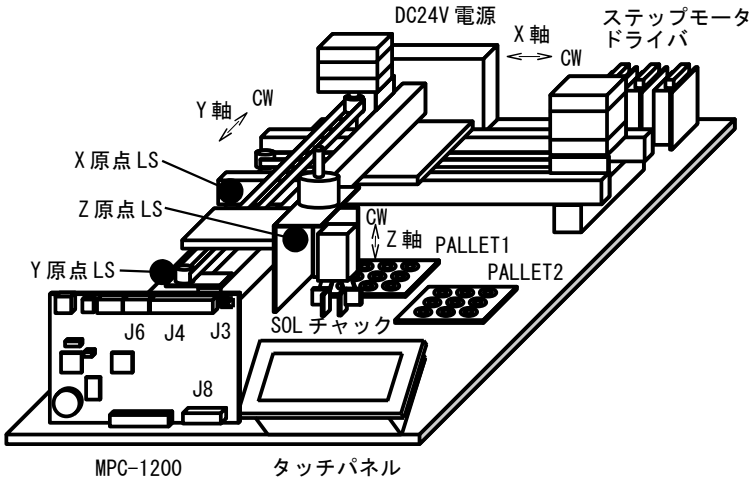
LAN・WAN 経由で MPC とデータ授受、コマンド制御ができます。MPC-IP はスレーブです。

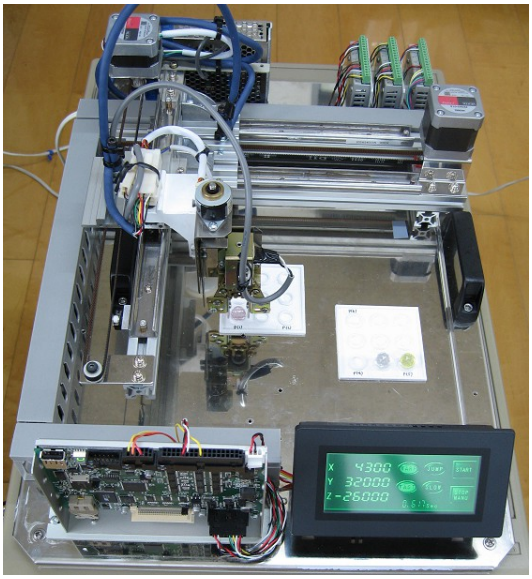


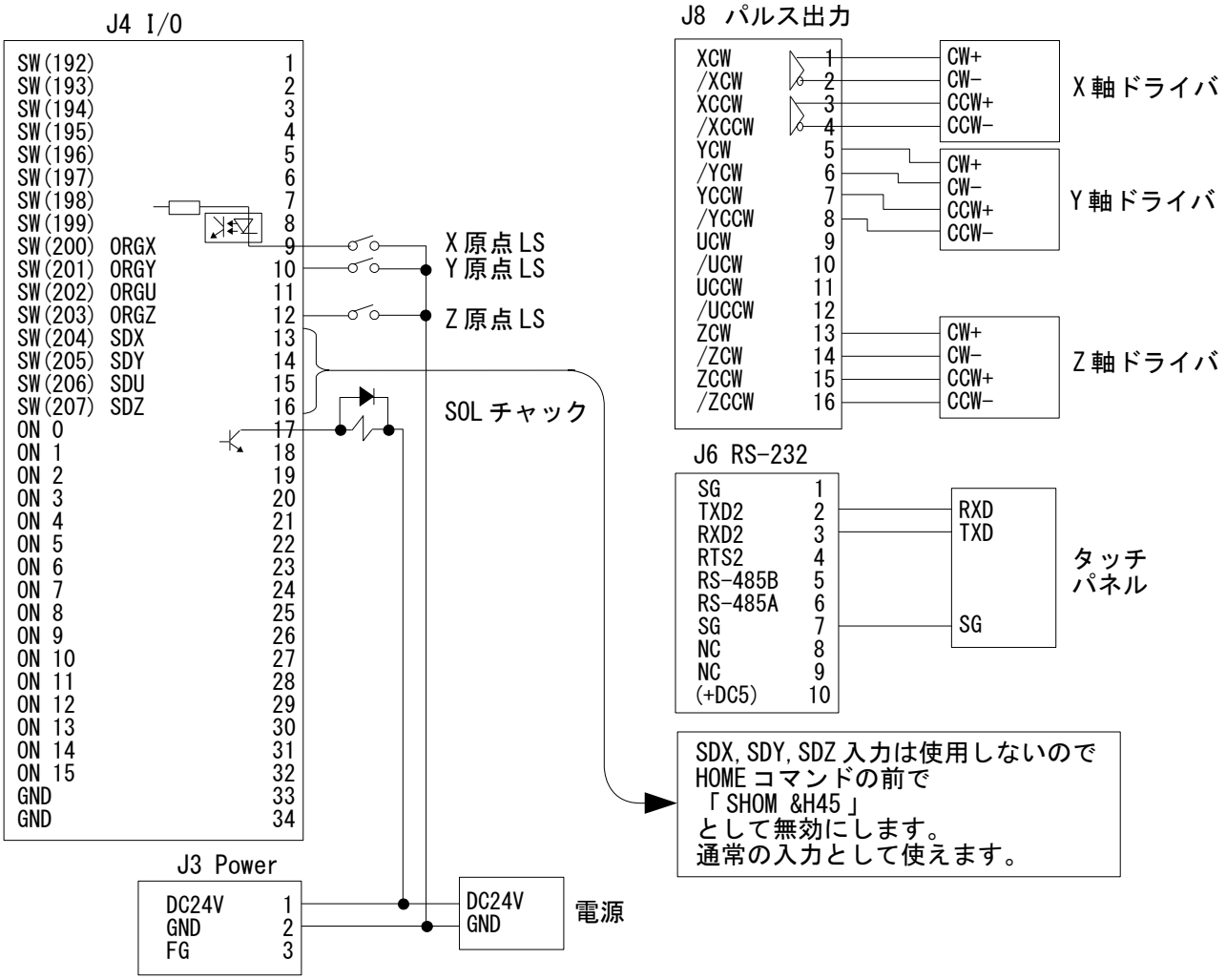
デモ機 XY14

XY14 はステップモーター・ベルト駆動の 3 軸直交ロボットのデモ機です。Z 軸のソレノイドチャックでワークをつかみ、パレット間の移載をします。
MPC-1200 のみで動作します(他の CPU ボートと MPG-2314 でも可能)。
パレットの作業点は各 3 箇所のティーチング点より PALLET コマンドで算出します。ティーチングはタッチパネルで行います。

機器構成と結線







タッチパネル画面

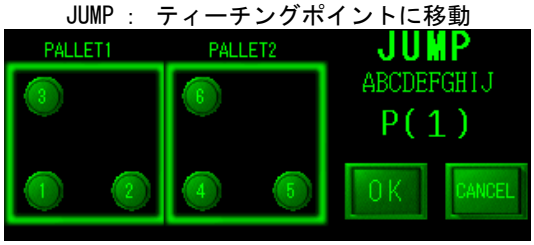
キーエンス VT3-W4G 、デザイナー VT STUDIO 上のイメージです。

- 自動運転、マニュアル時の画面。

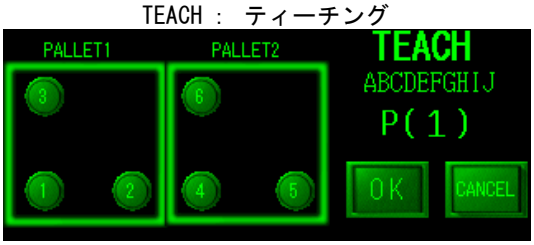
AUTO ： 自動運転

MANUAL ： JUG 移動、チャック開閉

JUMP ： ティーチングポイントに移動

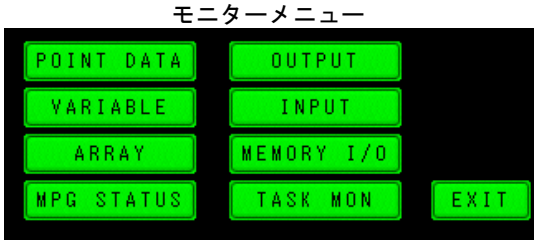


TEACH ： ティーチング



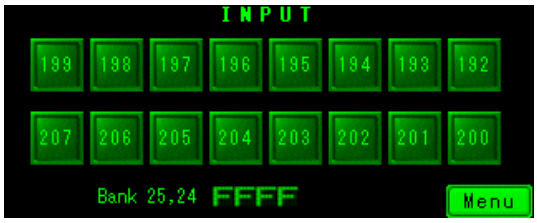
- タッチパネルの項でも紹介した各状況のモニター画面です。

モニターメニュー



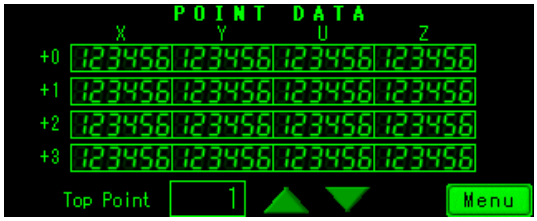
出力

入力



メモリーI/O

点データ



変数

配列変数

MPG 状態

タスク別行番号

プログラム

```
    PG 17                          /* MPC-1200は PG 17
    QUIT_FORK 1 *Main
    END                            /* タスク0 終了

  *Main
    PG 17                          /* このタスクにも PG 17 をアサイン
    MEWNET 38400 1                 /* パソコンアプリ用
    MEWNET 38400 2                 /* タッチパネル用
    MBK(4)=200                     /* タッチパネル 自動運転画面
    MBK(101)=100                   /* Default JOG Distance
    MBK(102)=1                     /* Default JUMP Point No
    MBK(103)=1                     /* Default TEACH Point No
    MBK(104)=0                     /* Default Error Number

    DO
      OFF 70000                    /* Start SW
      page=MBK(4)                  /* タッチパネルのページで動作を切り替える
      SELECT_CASE page
        CASE 200                   /* AUTO Page
          QUIT_FORK 2 *Auto        /* 自動モード
          QUIT_FORK 3 *Coordinates /* 座標表示
          QUIT_FORK 4 *SetWarp     /* WARP/JUMP切替
          DO
            WAIT MBK(4)<>page
            IF MBK(4)>=1000 THEN
              MBK(8140)=17
              WAIT MBK(4)<1000
            ELSE
              PULSE_OUT 2 0
              BREAK
            END_IF
          LOOP
        CASE 201                   /* JOG Page
        CASE 202                   /* JUMP Page
        CASE 203                   /* TEACH Page
          QUIT_FORK 2 *Manu        /* マニュアルモード
          WAIT MBK(4)<>page
        CASE 204                   /* ERROR Page
          WAIT MBK(4)<>page
        CASE_ELSE
      END_SELECT
      QUIT 2 3 4
      STOP ALL_A STP_D
      WAIT RR(ALL_A)==0
    LOOP

  *Manu                            /* マニュアル
    PG 17
    GOSUB *SetAccel
    FEED X_A|Y_A 10
    FEED Z_A 40
    S_MBK ″          ″ 210 10
    DO
      SELECT_CASE VOID
        CASE SW(70200)             /* X+ JOG移動
          RMVS X_A MBK(101)        /* X軸相対移動。移動量は MBK(101の値)
          WAIT RR(ALL_A)==0
          WAIT SW(70200)==0
        CASE SW(70201)             /* X-
```

```
        RMVS X_A MBK(101)*-1
        WAIT RR(ALL_A)==0
        WAIT SW(70201)==0
      CASE SW(70202)              /* Y+
        RMVS Y_A MBK(101)
        WAIT RR(ALL_A)==0
        WAIT SW(70202)==0
      CASE SW(70203)              /* Y-
        RMVS Y_A MBK(101)*-1
        WAIT RR(ALL_A)==0
        WAIT SW(70203)==0
      CASE SW(70204)              /* Z+
        RMVS Z_A MBK(101)
        WAIT RR(ALL_A)==0
        WAIT SW(70204)==0
      CASE SW(70205)              /* Z-
        RMVS Z_A MBK(101)*-1
        WAIT RR(ALL_A)==0
        WAIT SW(70205)==0
      CASE SW(70206)              /* HOMEボタン
        GOSUB *HOME
        WAIT SW(70206)==0
      CASE SW(70207)              /* JUMPボタン (Jump Page)
        LIMZ 0
        S_MBK "Jumping to" 210 10
        JUMP P(MBK(102))
        WAIT RR(ALL_A)==0
        S_MBK "           " 210 10
        MBK(103)=MBK(102)
        MBK(4)=201
      CASE SW(70208)              /* TEACHボタン (Teach Page)
        SETP MBK(103) P(0)
        FOR d=0 TO 1
          S_MBK "Completed " 210 10
          TIME 250
          S_MBK "          " 210 10
          TIME 250
        NEXT
        MBK(4)=201
      CASE SW(70209)              /* チャック開閉
        IF SW(0) THEN
          OFF 0
        ELSE
          ON 0
        END_IF
        WAIT SW(70209)==0
       CASE_ELSE
    END_SELECT
    SWAP
  LOOP

*Auto                             /* 自動運転
  PG 17

  /*     PALLET1                PALLET2
  /*
  /*     7    8    9           7    8    9
  /*     ◎    ○    ○           ◎    ○    ○
  /*    P(3)                   P(6)
  /*     4    5    6           4    5    6
```

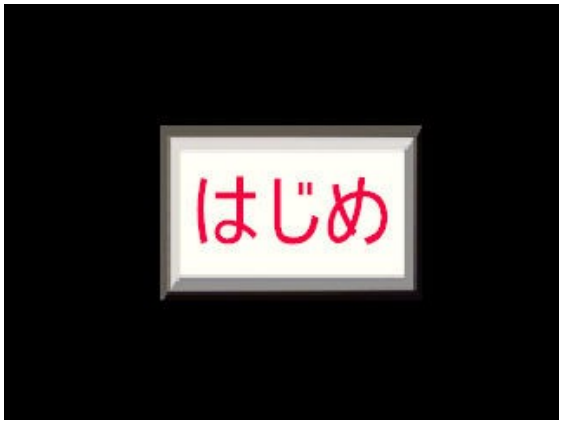
```
/*     ○   ○   ○        ○   ○   ○
/*
/* pln=1   2   3    pln=1   2   3
/*     ◎   ○   ◎        ◎   ○   ◎
/*    P(1)      P(2)     P(4)      P(5)

PALLET 1 P(1) P(2) P(3) 3 3   /* パレット上の3点から全マトリック計算
PALLET 2 P(4) P(5) P(6) 3 3
ChuckDelayO=10                /* Chuck Open Delay
ChuckDelayC=50                /* Chuck Close Delay
ChuckDelayW=200               /* Chuck Open/Close Delay
ON 70001                      /* WARP SW
ON 70002                      /* FAST SW
OFF 0                         /* Chuck

DO
   WAIT SW(70000)==1           /* Start SW
  GOSUB *HOME
  DO
    WAIT SW(70000)==1
    GOSUB *SetAccel
    MOVS X_A|Y_A P(1)
    WAIT RR(ALL_A)==0
    Iz=Z(1)+18000
    LIMZ Iz
    GOSUB *Pallet1to2 1 3
    GOSUB *Pallet2to1 1 3 3
    GOSUB *Pallet1to2 4 6
    GOSUB *Pallet2to1 4 6 3
    GOSUB *Pallet1to2 7 9
    GOSUB *Pallet2to1 7 9 -6
    MOVS Z_A 0
    WAIT RR(ALL_A)==0
    FEED ALL_A fed
    MOVS 50000 0 0 0          /* 周回動作(デモ:特に意味無し)
    WAIT RR(ALL_A)==0
    MOVS 50000 50000 0 0
    WAIT RR(ALL_A)==0
    MOVS 0 50000 0 0
    WAIT RR(ALL_A)==0
    MOVS 0 1000 0 0
    WAIT RR(ALL_A)==0
    MOVS 0 50000 0 0
    WAIT RR(ALL_A)==0
    MOVS 50000 50000 0 0
    WAIT RR(ALL_A)==0
    MOVS 50000 0 0 0
    WAIT RR(ALL_A)==0
    MOVS 0 0 0 0
    WAIT RR(ALL_A)==0
    TIME 100
  LOOP
LOOP

*Pallet1to2                    /* PALLET1 -> PALLET2
  _VAR sp_ ep_
  FOR pln=sp_ TO ep_
    LIMZ Iz
    FEED ALL_A fed
    WARP up_z1 PL(1;pln) dwn_Z1 /* PALLET1に取りに行く
```

```
    WAIT RR(ALL_A)==0
    WAIT SW(70000)==1           /* SW(70000)==0 ならPAUSE
    ON 0                        /* チャック閉
    TIME ChuckDelayC
    stclk=SYSCLK
    FEED ALL_A fed
    LIMZ Z(1)+4000
    WARP up_z2 PL(2;pln) dwn_Z2 /* PALLET2に置きに行く
    WAIT RR(ALL_A)==0
    OFF 0                       /* チャック開
    MBK(300)=SYSCLK-stclk
    TIME ChuckDelay0
    WAIT SW(70000)==1           /* SW(70000)==0 ならPAUSE
  NEXT
  RETURN
*Pallet2to1                     /* PALLET2 -> PALLET1
  _VAR sp_ ep_ of_
  FOR pln=sp_ TO ep_ /* STEP -1
    LIMZ lz
    FEED ALL_A fed
    WARP up_z1 PL(2;pln) dwn_Z1 /* PALLET2に取りに行く
    WAIT RR(ALL_A)==0
    WAIT SW(70000)==1
    ON 0
    TIME ChuckDelayC
    stclk=SYSCLK
    FEED ALL_A fed
    LIMZ Z(1)+4000
    WARP up_z2 PL(1;pln+of_) dwn_Z2 /* PALLET1に置きに行く
    WAIT RR(ALL_A)==0
    OFF 0
    MBK(300)=SYSCLK-stclk
    TIME ChuckDelay0
    WAIT SW(70000)==1
  NEXT
  RETURN

*HOME                           /* 原点復帰
  ACCEL X_A|Y_A 10000 1000 500
  ACCEL Z_A 8000 500 100
  SHOM &H45                     /* SDZ,SDY,SDX入力無効
  IF SW(203)==1 THEN            /* Z LSがオンなら退避移動
    RMVS Z_A -3000
    WAIT RR(Z_A)==0
  END_IF
  HOME Z_A POS_L                /* Z軸原点復帰
  WAIT RR(ALL_A)==0
  IF SW(200)==1 THEN            /* X LSがオンなら退避移動
    RMVS X_A 5000
  END_IF
  IF SW(201)==1 THEN            /* Y LSがオンなら退避移動
    RMVS Y_A 5000
  END_IF
  WAIT RR(ALL_A)==0
  TIME 500
  HOME X_A|Y_A NEG_L            /* XY軸原点復帰
  WAIT RR(ALL_A)==0
  RMVS X_A|Y_A 1000             /* OFF SET
  WAIT RR(ALL_A)==0
  CLRPOS
  TIME 500
```

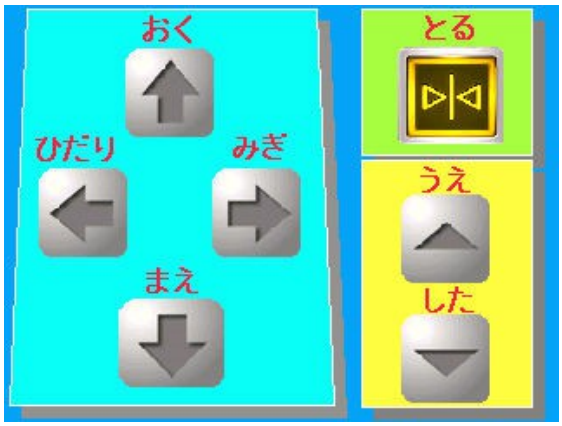
```
  RETURN

*SetAccel                       /* 加減速、最高速設定
  ACCEL X_A|Y_A 120000 12000 500
  ACCEL Z_A X(10) X(10)/20 100
  RETURN

*SetWarp                        /* WARP/JUMP切替
  PG 17
  ON 70002                      /* FAST/SLOW
  DO
    IF SW(70001) THEN           /* WARP/JUMPスイッチ
      up_z1=11000               /* 取りに行く時の上昇位置  置いたワークの上まで上がる
      dwn_Z1=11000              /* 取りに行く時の下降位置  取るワークの上から入る
      up_z2=2000                /* 置きに行く時の上昇位置
      dwn_Z2=2000               /* 置きに行く時の下降位置
      yoko_z=10000
    ELSE
      up_z1=0
      up_z2=0
      dwn_Z1=0
      dwn_Z2=0
      yoko_z=0
    END_IF
    IF SW(70002) THEN           /* FAST/SLOWスイッチ
      fed=100
    ELSE
      fed=50
    END_IF
    SWAP
  LOOP

*Coordinates                    /* タッチパネル座標表示
  PG 17
  DO
    MBK(200~Lng)=X(0)
    MBK(202~Lng)=Y(0)
    MBK(204~Lng)=Z(0)
    SWAP
  LOOP
```

あめちゃんキャッチャー

　XY14 を使ったクレーンゲームです。MPC-1200 単体で動作、タッチパネルで操作します。通常のクレーンゲームと違い、Z 軸も自由に上下して何度でも位置を修正できます。
(諏訪圏工業メッセ 2015 に出展。子供に人気です)

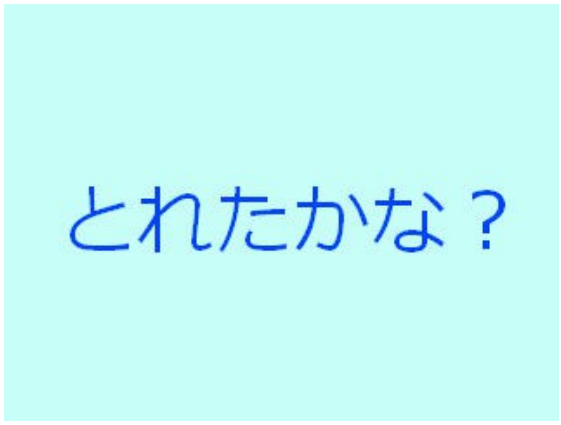


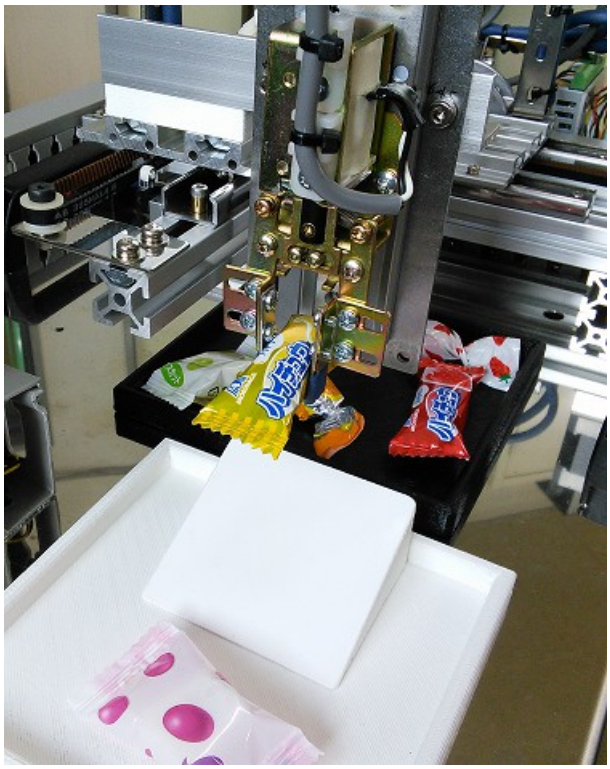
「はじめ」ボタンを押してゲーム開始



方向ボタンを押している間連続で動作します。
（エリア外には動きません）

「とる」ボタンでつまみ上げて手元へ運びます。





プログラム

```
  PG 17
  QUIT_FORK 1 *Main
  END

*Main
  MEWNET 38400 2                  /* タッチパネル
  PageCtrl=8                      /* GP-4301のページ切替アドレス
  StartPage=1                     /* "はじめ" のページ
  PlayPage=5                      /* 水平垂直操作ページ
  ConfirmPage=9                   /* "とれたかな" のページ
  OFF 0

  MBK(PageCtrl)=StartPage
  PG 17                           /* MPC-1200パルス出力
  SETP 10 54000 2000 0 0    /* 排出点
  /* 可動エリア設定
  /*                  X2,Y2
  /*       +-----------+
  /*       |           |       + Z1 Up
  /*       |           |       |
  /*       |           |       |
  /*       |           |       + Z2 Down
  /*       +-----------+
  /*     X1,Y1
  /*    + 0,0
  X1=27500                        /* X下限
  Y1=18000                        /* Y下限
  X2=55500                        /* X上限
  Y2=51000                        /* Y上限
  Z1=0                            /* Z上限
  Z2=-47900                       /* Z下限

  DO
    GOSUB *HOME
    GOSUB *SetAccel
    MOVL X1 Y1
    WAIT RR(ALL_A)==0
    QUIT_FORK 2 *DRIFT            /* "はじめ"ボタンが押されるまで上を旋回
    MBK(PageCtrl)=StartPage
    WAIT SW(70000)==1             /* "はじめ"ボタン
    QUIT 2
    STOP ALL_A STP_I
    WAIT RR(ALL_A)==0
    MBK(PageCtrl)=PlayPage
    OFF 2
    DO
      SELECT_CASE VOID
        CASE SW(70200)            /* "みぎ"ボタン
          swno=70200              /* ボタン番号
          axis=X_A                /* X軸
          area=X2                 /* CW方向稼動範囲
          direction=1             /* CW方向
          GOSUB *MOVING
        CASE SW(70201)            /* "ひだり"ボタン
          swno=70201
          axis=X_A
          area=X1
          direction=-1
          GOSUB *MOVING
        CASE SW(70202)            /* "おく"ボタン
```

```
            swno=70202
            axis=Y_A
            area=Y2
            direction=1
            GOSUB *MOVING
          CASE SW(70203)          /* "まえ"ボタン
            swno=70203
            axis=Y_A
            area=Y1
            direction=-1
            GOSUB *MOVING
          CASE SW(70204)          /* "うえ"ボタン
            swno=70204
            axis=Z_A
            area=Z1
            direction=1
            GOSUB *MOVING
          CASE SW(70205)          /* "した"ボタン
            swno=70205
            axis=Z_A
            area=Z2
            direction=-1
            GOSUB *MOVING
          CASE SW(70209)          /* "とる"ボタン
            GOSUB *PICKUP
            BREAK
          CASE_ELSE
        END_SELECT
        SWAP                      /* タスクスワップ
      LOOP
    LOOP

*MOVING                           /* 左右前後上下の動作
    QUIT 2
    STOP ALL_A
    WAIT RR(ALL_A)==0
    QUIT_FORK 2 *PULSE_OUT        /* 別タスクでパルスを出し、エリアとタッチパネルボタンを監視
    WAIT RR(ALL_A)<>0             /* 移動開始待ち
    DO                            /* ボタンから指が離れたりエリア外に出たら停止
      IF axis==X_A THEN
          currentpos=X(0)         /* X軸現在位置取得
      END_IF
      IF axis==Y_A THEN
          currentpos=Y(0)         /* Y軸現在位置取得
      END_IF
      IF axis==Z_A THEN
          currentpos=Z(0)         /* Z軸現在位置取得
      END_IF

      IF (currentpos>=area)&(direction==1) THEN   /* 移動範囲チェック
        STOP axis STP_I                           /* 範囲外なら停止
      END_IF
      IF (currentpos<=area)&(direction==-1) THEN  /* 移動範囲チェック
        STOP axis STP_I
      END_IF
      IF SW(swno)==0 THEN         /* ボタンから指が離れたら停止
        STOP axis STP_I
        OFF 2
        BREAK                     /* Exit from DO LOOP
      END_IF
```

```
    SWAP
  LOOP
  RETURN

*PULSE_OUT
  PG 17
  FEED X_A|Y_A 25
  FEED Z_A 50
  RMVC axis direction           /* 移動
  IF (axis==Z_A)&(direction==1) THEN      /* Z軸上方向なら原点LSを監視
    DO
      IF (RR(ALL_A)==0)|(SW(203)) THEN    /* *MOVING で停止またはZ原点がオン
        STOP axis STP_I
        IF SW(203)==1 THEN
          CLRPOS Z_A
        END_IF
        BREAK
      END_IF
      SWAP
    LOOP
  END_IF
  WAIT RR(ALL_A)==0             /* パルス停止待ち
  END                           /* 自己タスク終了

*PICKUP                         /* 飴を持ち上げる
  QUIT 2
  STOP ALL_A
  WAIT RR(ALL_A)==0
  FEED Z_A 50
  MOVS Z_A Z2                   /* Z軸下降
  WAIT RR(Z_A)==0
  MBK(PageCtrl)=ConfirmPage
  ON 0                          /* チャック閉
  TIME 500
  FEED ALL_A 70
  HOME Z_A POS_L                /* 原点LSオンまで上昇
  WAIT RR(ALL_A)==0
  FEED ALL_A 100
  MOVS P(10)                    /* 排出点
  WAIT RR(ALL_A)==0
  OFF 0                         /* チャック開
  TIME 500
  MOVS 4000 4000 0 -4000
  WAIT RR(ALL_A)==0
  RETURN

*DRIFT                          /* 稼動範囲四隅を移動
  PG 17
  FEED X_A|Y_A 50
  FEED Z_A 20
  DO
    MOVS X1 Y1 0 0
    WAIT RR(ALL_A)==0
    TIME 100
    MOVS X1 Y2 0 0
    WAIT RR(ALL_A)==0
    TIME 100
    MOVS X2 Y2 0 0
    WAIT RR(ALL_A)==0
    TIME 100
```

```
    MOVS X2 Y1 0 0
    WAIT RR(ALL_A)==0
    TIME 100
  LOOP

*HOME                              /* 原点復帰
  ACCEL X_A|Y_A 10000 2000 500
  ACCEL Z_A 15000 500 100
  SHOM &H45                        /* SDZ,SDY,SDX入力無効
  IF SW(203)==1 THEN               /* Z原点LSがオンなら退避移動
    RMVS Z_A -5000
    WAIT RR(Z_A)==0
  END_IF
  HOME Z_A POS_L                   /* 原点復帰
  WAIT RR(ALL_A)==0
  RMVS Z_A -3000                   /* OFF SET
  WAIT RR(ALL_A)==0
  IF SW(200)==1 THEN               /* X原点LSがオンなら退避移動
    RMVS X_A 5000
  END_IF
  IF SW(201)==1 THEN               /* Y原点LSがオンなら退避移動
    RMVS Y_A 5000
  END_IF
  WAIT RR(ALL_A)==0
  TIME 500
  HOME X_A|Y_A NEG_L               /* XY原点復帰
  WAIT RR(ALL_A)==0
  CLRPOS
  TIME 500
  RETURN

*SetAccel                          /* 加減速、最高速設定
  xyaccel=30000
  zaccel=30000
  ACCEL X_A|Y_A xyaccel 12000 500
  ACCEL Z_A zaccel 3000 100
  RETURN
```