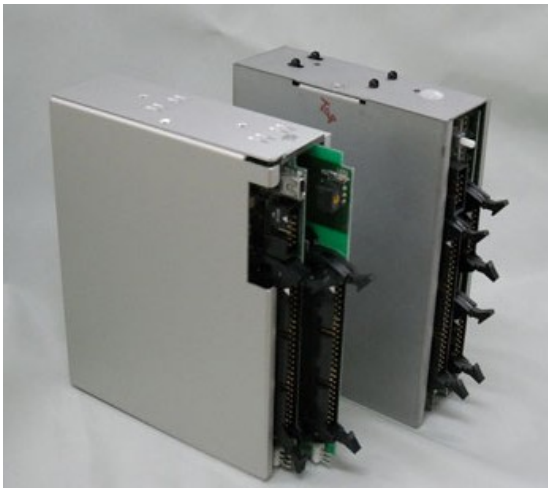
MPC-816 から MPC-N816 への移行資料

2011/09/13

MPC-N816 と MPC-816 の概要

- システム構成例

MPC-N816	MPC-816
◆CPU 1 枚 MPC-N816 + CASE-1S (入力 16、出力 8) ◆CPU+I/O MPC-N816 + MIO-N816 + CASE-2S (入力 32、出力 16) ◆CPU+PG MPC-N816 + MPG-2541 又は MPG-2314 + CASE-2S (入力 16、出力 8、パルス 4 軸) ◆CPU+AD/DA MPC-N816 + MPC-AD12 + CASE-2S (入力 16、出力 8、AD 入力 8、DA 出力 4) ◆ラック(注意) RACK-V4S：4 スロット RACK-8S：8 スロット (RACK-16S：16 スロット) (注意)ラック使用時は MPC-N816 内部給電容量(800mA)を超えないように注意してください。MPC-N816 は 4 スロットラック程度の小規模向きです。 ※出力点数に MPC-N816・J6 コネクタの出力 8 点は含まれていません。	◆CPU 1 枚 MPC-816XC (MPC-816 + ケース B) ◆CPU+I/O MPC-SET(EX) (MPC-816 ＋ MIF-816 + ケース E) ◆CPU+PG MPC-SET(303) (MPC-816 ＋ MPG-303 + ケース E) ◆CPU+AD/DA MPC-SET(AD) (MPC-816 ＋ MIF-816AD + ケース E) ◆ラック RACK-SET(A)： 4 スロット RACK-SET(B)： 8 スロット RACK-SET(C)： 16 スロット
・MPC-N816 と MPC-816 のケース、ラック、拡張ボードに互換性はありません。混載できません。 ・MPC-N816 のケース、ラック、拡張ボードは MPC-2000 シリーズと共通です。 ・プログラミング言語、プログラム開発環境(ケーブル、ターミナルソフト、システムローダ、エディタ等)も MPC-2000 シリーズと共通です。	



奥 MPC-816、MPG-303、ケース-E
手前 MPC-N816、MIO-N816、CASE-2S
(写真は実験用のものです)

奥 MPC-816、MIF-816、MPG-303、RACK-SET(A)
手前 MPC-N816 、MIO-N816 、MPG-2541、RACK-V4S
取り付け寸法は互換性あります。

- CPU ボード 機能

MPC-N816	MPC-816
◆J1 コネクタ (10 ピン PS コネクタ) ユーザー RS-232：1Ch, Max38400bps プログラム RS-232：1Ch, 38400,pn,8bit,s1,none 固定 RS-485：1Ch, Max 38400bps ユーザー RS-232 は MEWNET コマンドでタッチパネルと接続可(通常の RS-232 機能とは併用できません) ◆J4 コネクタ(ロック付き 50 ピン PS コネククタ) 入力：フォトカプラ 16 点 アンプ内臓 2 線式センサ対応(漏れ電流 1.5mA 以下) 出力：オープンコレクタ 8 点 ◆J6 コネクタ(16 ピン PS コネクタ) 出力：C-MOS 8 点 (パルス出力、PWM 出力) ◆J7 コネクタ(mini-A USB コネクタ) USB メモリ ◆J3 コネクタ(JST 3 ピン) DC24V 電源給電	◆J1 コネクタ(10 ピン PS コネクタ) ユーザー RS-232：1Ch, Max19200bps プログラム RS-232：1Ch, 9600,pe,8bit,s1,none 固定 ◆J4 コネクタ(ロック付き 50 ピン PS コネククタ) 入力：フォトカプラ 16 点 出力：オープンコレクタ 8 点 ◆J3 コネクタ(JST 4 ピン) DC24V 電源給電

- I/O ボード

MPC-N816	MPC-816
◆MIO-N816 (ロック付き 50 ピン PS コネクタ) MPC-816 と同仕様のピン配列 入力 16/出力 8 アンプ内臓 2 線式センサ対応(漏れ電流 1.5mA 以下) そのほか MPC-2000 シリーズ I/O ボードが使用可能 MIO-1616 ：入力 16,出力 16 MIO-3232 ：入力 32,出力 32 MIP-0064 ：入力 64 MOP-0064 ：出力 64	◆MIF-816 拡張 I/F ボード。入力 16,出力 8 ◆MIO-816 入力 16,出力 8 ◆MIO-248 入力 8,出力 24

- パルス発生ボード

MPC-N816	MPC-816
◆MPG-2314 4 軸、Max4Mpps、直線補間、円弧補間、S 字加減速 カウンタ入力 ◆MPG-2541 4 軸、Max400kpps、S 字加減速対応、補間機能無し。 簡易的なパルス発生ボードで、制約事項があります。 マルチタスク使用、現在点読み込み・変更等注意。 PG コマンドで任意のタスクから MPG ボードを制御できます。 異なるタスクで同一ボード上の異なる軸を制御できます。	◆MPG-303 4 軸 Max60kpps XY/UZ 排他 2 軸直線補間 MPG 制御タスクはあらかじめ決まっています。 TASK0 ： PG コマンド設定 TASK1～3：MPG-303#1 TASK4～7：MPG-303#2 TASK7～11：MPG-303#3

- 通信ボード

MPC-N816	MPC-816
◆MRS-MCOM RS-232 専用：1Ch RS-232,422,485 兼用：2Ch タッチパネル MEWNET 対応 ◆MPC-CUnet2 MPC～MPC、MPC～PC メモリ共有 PC 側は USB-CUnet	◆MBK-816 タッチパネル MEWNET 対応 RS-232、485 拡張ボード無し

- AD、DA ボード

MPC-N816	MPC-816
◆MPC-AD12 AD 入力：12bit、8 点 DA 出力：12bit、4 点	◆MIF-816AD AD 入力：12bit、4 点 DA 出力：12bit、1 点

- タスク数、プログラム容量、ファイル名、変数・定数

MPC-N816	MPC-816
◆タスク数 32 ◆プログラム容量 155Kbyte(1 行 25byte として約 6200 行) ◆パソコン上のファイル拡張子 プログラム *.F2K、点データ *.P2K ◆変数・定数 変数： 変数名 15 文字以内 最大 2000 個 タスクローカル変数： 各タスク 32 個 文字列変数： 128 個。文字列長 255 以下 配列変数： DIM コマンドで宣言、全 20000 個 点データ： P(1)～P(7000) 予約変数： ptr_ , rse_ , SYSCLK, SEC など 予約定数： ALL_A, X_A, VOID など 予約変数・定数は定義通りに記述します。 例えば、ALL_A は○、 All_a は× ◆ラベル文字数 max100 文字(半角)	◆タスク数 12 (P 版) ◆プログラム容量 2044 行 ◆パソコン上のファイル拡張子 プログラム *.FSC、点データ *.PSC ◆変数 変数：A～Z、A0～Z9 固定 配列変数：AR(0)～AR(31)、M(0)～M(5999) 点データ：P(1)～P(300)

- 演算子

	MPC-N816	MPC-816
加	+	+
減	-	-
乗	*	*
除	/	/
余	%	%
論理積(AND)	&	^
論理和(OR)	|	|
排他的論理和(XOR)	^	x
左シフト	<<	
右シフト	>>	
ワード合成	,	
上位バイト	;	
データ長	4 バイト整数	3 バイト整数(P 版)

- その他

・MPC-N816 に S-RAM バックアップバッテリはありません。点データの一部はフラッシュ ROM 保存が可能です。
・USB メモリに点データやテキストデータ(TXT、CSV 等)の保存、読み込みができます。
・入力 192～198 は AD 入力(分解能 10bit)としてもしても使用できます(通常の入力と排他的使用)。

プログラム例

- プログラム方法 (1)

MPC-N816 も MPC-816 と同様に FTMW で直接コマンド実行、プログラミング、デバッグ、保存等が行えます。
直接プログラムする場合は文番号を付けて入力します。MPC-816 では FORK,GOTO などの飛び先を文番号で指定しすることができますが、MPC-N816 は全てラベルで記述します。
実行は RUN コマンドです。(自動実行はプログラミングケーブルを抜いて電源オン。)
停止は Ctrl+A です。

MPC-N816	MPC-816
#NEW #10 FORK 1 *TASK1 20 *LOOP 30 ON 0 40 TIME 100 50 OFF 0 60 TIME 100 70 GOTO *LOOP 100 *TASK1 110 ON 1 120 TIME 500 130 OFF 1 140 TIME 500 150 GOTO *TASK1 #RUN ←実行 52* ←編集後 1 回目 FROM 書き込みメッセージ Compiling 2 Labels -Pass_1 completed -Pass_2 completed -Pass_3 completed -GetPrgSum 4D7D010D ------ *0 [60] *1 [140] ←Ctrl+A で停止 # 各タスクの停止行表示	>NEW >10 FORK 1 100 20 *LOOP 30 ON 0 40 TIME 10 50 OFF 0 60 TIME 10 70 GOTO *LOOP 100 *TASK1 110 ON 1 120 TIME 50 130 OFF 1 140 TIME 50 150 GOTO *TASK1 RUN Programming the FLASH ROM *++++++ TASK 0 # 60 TASK 1 # 120 >

- プログラム方法 (2)

エディタで作成して読み込む(LOAD)、PC に保存する(SAVE)。実行・停止は上記(1)と同じです。
文番号を付けずにコーディングします。「’」の後ろにコメントが書けます。
MPC-N816 は 「/*」 の後ろにソースコメント、MPC-816 は「”」の後ろにソースコメントが書けます。
なお、以降のサンプルはエディタで作成して LOAD、RUN したものです。

MPC-N816	MPC-816
◆ソースプログラム (*.f2k) '---------- 'MAIN TASK '---------- FORK 1 *TASK1 /* sub-task1 execute *LOOP ON 0 /* output #0 turn on TIME 100 /* delay 0.1sec OFF 0 /* output #0 turn off TIME 100 GOTO *LOOP '---------- 'TASK1 '---------- *TASK1 ON 1 TIME 500 OFF 1 TIME 500 GOTO *LOOP ◆↓ 読み込み(LOAD) #LIST	◆ソースプログラム (*.fsc) '---------- 'MAIN TASK '---------- FORK 1 *TASK1 ” sub-task1 execute *LOOP ON 0 ” output #0 turn on TIME 10 ” delay 0.1sec OFF 0 ” output #0 turn off TIME 10 GOTO *LOOP '---------- 'TASK1 '---------- *TASK1 ON 1 TIME 50 OFF 1 TIME 50 GOTO *TASK1 ◆↓ 読み込み(LOAD) >LIST

```
10      '----------
20      'MAIN TASK
30      '----------
40        FORK   1 *TASK1
50      *LOOP
60        ON   0
70        TIME   100
80        OFF   0
90        TIME   100
100       GOTO  *LOOP
110     '----------
120     'TASK1
130     '----------
140     *TASK1
150       ON   1
160       TIME   500
170       OFF   1
180       TIME   500
190       GOTO  *LOOP
#
```

◆↓保存(SAVE)
(注意)同名で保存すると /*コメント が無くなる

```
'----------
'MAIN TASK
'----------
  FORK   1 *TASK1
*LOOP
  ON   0
  TIME   100
  OFF   0
  TIME   100
  GOTO  *LOOP
'----------
'TASK1
'----------
*TASK1
  ON   1
  TIME   500
  OFF   1
  TIME   500
  GOTO  *LOOP
```

```
10      '----------
20      'MAIN TASK
30      '----------
40      FORK 1,*TASK1
50      *LOOP
60      ON 0
70      TIME 10
80      OFF 0
90      TIME 10
100     GOTO *LOOP
110     '----------
120     'TASK1
130     '----------
140     *TASK1
150     ON 1
160     TIME 50
170     OFF 1
180     TIME 50
190     GOTO *TASK1
>
```

◆↓ 保存(SAVE)
(注意)同名で保存すると ″コメント が無くなる

```
  '----------
  'MAIN TASK
  '----------
  FORK 1,*TASK1
*LOOP
  ON 0
  TIME 10
  OFF 0
  TIME 10
  GOTO *LOOP
  '----------
  'TASK1
  '----------
*TASK1
  ON 1
  TIME 50
  OFF 1
  TIME 50
  GOTO *TASK1
```

- I/O (1)

入力、出力制御は似ています。

MPC-N816

```
10        DO                          ← DO～LOOP 繰り返し
20         ON   0                     ← bit オン
30         TIME   100
40         OFF   0
50         TIME   100
60         IF   SW(192)==1 THEN       ← bit 入力
70          GOSUB   *SW_STAT
80         END_IF
90        LOOP
100     *SW_STAT
110       PRINT   ″SW(192)″ SW(192)
120       WAIT   SW(192)==0           ← bitオフ待ち
130       RETURN
RUN

 SW(192) 1
 SW(192) 1
 SW(192) 1
```

MPC-816

```
10      *LOOP
20      ON 0
30      TIME 10
40      OFF 0
50      TIME 10
60      IF SW(0)=1 GOSUB *SW_STAT
70      GOTO *LOOP
80      *SW_STAT
90      'SW(0)
100     PRINT STR(-1),SW(0)
110     WAIT SW(0)=0
120     RETURN
>RUN
 SW(0) 1
 SW(0) 1
 SW(0) 1
```

- I/O (2)

DSW 読み込み。

このサンプルの DSW は MPC-N816 の IN(24)、MPC-816 の IN(0)の上位 4 ビットに接続されています。

MPC-N816

```
10         DO
20        DSW=IN(24)/16          ← 8bit 読んで 4bit 下位にシフト
30          PRINT   DSW
40          WAIT   DSW<>(IN(24)/16)
50          TIME   100
60         LOOP
#RUN

 0
 1
 2
 3
 4
 5
 6
```

MPC-816

```
10       *LOOP
20        DO=IN(0)/16
30       PRINT DO
50       *LOOP1
60        D1=IN(0)/16
70       IF D1=DO THEN *LOOP1
80       TIME 10
90       GOTO *LOOP
>RUN
 0
 1
 2
 3
 4
 5
 6
```

- サブルーチン(1)

GOSUB はネストが深すぎると Stack overflow エラーになります

MPC-N816

```
10        j=0
20         DO
30          GOSUB   *PASS
40       *PASS
50         j=j+1
60          PRINT   j
70         LOOP                    ← RETURN が無い
#RUN

 1
 2
  (中略)
 31
 32

[30]    Stack overflow(gosub-return):10
```

MPC-816

```
10        J=0
20       *LOOP
30       GOSUB *PASS
40       *PASS
50        J=J+1
60       PRINT J
70       GOTO *LOOP
RUN
 1
 2
  (中略)
  12
 13
 # 30
!! Stack Overflow
```

- サブルーチン(2)

MPC-N816 は GOSUB に引数、RETURN に戻り値を与えられます。(文字列は不可)
タスクローカル変数を使えば複数のタスクでサブルーチン共有が可能です。

MPC-N816

```
10         FORK   1 *TASK1
20         FORK   2 *TASK2
30         END
40       *TASK1
50         j_=0                    ← 末尾に「_」付きはタスクローカル変数
60         FOR   i_=0 TO 10
70          GOSUB   *INC i_ j_     ← サブルーチンの引数 i_ j_
80          _VAR   j_              ← サブルーチンの戻り値
90          SWAP
100        NEXT
110        END
120      *TASK2
130        j_=100
140        FOR   i_=0 TO 10
150         GOSUB   *INC i_ j_
160         _VAR   j_
170         SWAP
180        NEXT
190        END
200      *INC
210         _VAR   arg1_ arg2_     ← 引数受け取り
220        k_=arg1_+arg2_
230         WAIT   ON(-1)==0
```

MPC-816

None

```
240       PRINT   "TASK" TASKn ":" arg1_ "+" arg2_ "=" k_
250       OFF   -1
260       RETURN   k_              ← 戻り値 k_
#RUN

# TASK 1 : 0 + 0 = 0
 TASK 2 : 0 + 100 = 100
 TASK 1 : 1 + 0 = 1
 TASK 2 : 1 + 100 = 101
 TASK 1 : 2 + 1 = 3
 TASK 2 : 2 + 101 = 103
 TASK 1 : 3 + 3 = 6
 TASK 2 : 3 + 103 = 106
 TASK 1 : 4 + 6 = 10
 TASK 2 : 4 + 106 = 110
```

- FOR～NEXT

MPC-N816 は FOR～NEXT から途中離脱できます。MPC-816 は FOR~NEXT のネストが深すぎると Stack Overflow になります。

MPC-N816

```
0       j=0
20        DO
30          FOR   i=0 TO 10
40            GOTO  *PASS                ←FOR～NEXT から出る
50          NEXT
60      *PASS
70         j=j+1
80          IF   j%10000==0 THEN  : PRINT   j : END_IF
90          SWAP
100       LOOP
#RUN

 10000
 20000
 30000
 40000
```

MPC-816

```
10        J=0
20      *LOOP
30      FOR I=0 TO 10
40       GOTO *PASS
50      NEXT I
60      *PASS
70       J=J+1
80      PRINT J
90      GOTO *LOOP
>RUN
 1
 2
 (中略)
 8
 9
 # 30
!! Stack Overflow
```

- 条件分岐(1)

IF を使った MPC-816 風の条件分岐です。

MPC-N816

```
10      *LOOP
20        IF   SW(192)==1 THEN  : GOTO  *CASE0 : END_IF
30        IF   SW(193)==1 THEN  : GOTO  *CASE1 : END_IF
40        IF   SW(194)==1 THEN  : GOTO  *CASE2 : END_IF
50        SWAP
60        GOTO  *LOOP
70      *CASE0
80        PRINT   "SW192 is ON"
90        GOTO  *CASE_END
100     *CASE1
110       PRINT   "SW193 is ON"
120       GOTO  *CASE_END
130     *CASE2
140       PRINT   "SW194 is ON"
150       GOTO  *CASE_END
160     *CASE_END
170       TIME   100
180       WAIT   IN(24)&&H00000007==0
190       GOTO  *LOOP
#RUN

 SW192 is ON
 SW193 is ON
 SW194 is ON
```

MPC-816

```
10      *LOOP
20      IF SW(0)=1 THEN *CASE0
30      IF SW(1)=1 THEN *CASE1
40      IF SW(2)=1 THEN *CASE2
50      GOTO *LOOP
60      *CASE0
70      PRINT STR(-1)
80      GOTO *LOOP1
90      *CASE1
100     PRINT STR(-1)
110     GOTO *LOOP1
120     *CASE2
130     PRINT STR(-1)
140     GOTO *LOOP1
150     *LOOP1
160     TIME 10
170      A=IN(0)^&H07
180     IF A<>0 THEN *LOOP1
190     GOTO *LOOP
>RUN
 CASE0
 CASE1
 CASE2
```

- 条件分岐(2)

SELECT_CASE で変数の値をチェックしています。

MPC-N816	MPC-816
10　　DO 20　　 DO 30　　instatus=IN(24)&&H00000007 40　　 IF　instatus<>0 THEN 50　　 BREAK 60　　 END_IF 70　　 SWAP 80　　 LOOP 90　　 SELECT_CASE　instatus　　←変数の値で条件分岐 100　　 CASE　&H00000001 : PRINT　"SW192 is ON" 110　　 CASE　&H00000002 : PRINT　"SW193 is ON" 120　　 CASE　&H00000004 : PRINT　"SW194 is ON" 130　　 CASE_ELSE　: PRINT　"?" 140　　END_SELECT 150　　 TIME　100 160　　 WAIT　IN(24)&&H00000007==0 170　　LOOP #RUN SW192 is ON SW193 is ON SW194 is ON	None

- 条件分岐(3)

SELECT_CASE に VOID を与えて CASE に条件(式)を与えます。

MPC-N816	MPC-816
10　　DO 20　　 WAIT　SW(192)|SW(193)|SW(194) 30　　 SELECT_CASE　VOID　　← VOID 指定 40　　 CASE　SW(192) : PRINT　"SW(192) ON" 50　　 CASE　SW(193) : PRINT　"SW(193) ON" 60　　 CASE　SW(194) : PRINT　"SW(194) ON" 70　　 CASE_ELSE　: PRINT　"?" 80　　END_SELECT 90　　 TIME　100 100　　 WAIT　(SW(192)==0)&(SW(193)==0)&(SW(194)==0) 110　　LOOP #RUN SW(192) ON SW(193) ON SW(194) ON	None

- RS-232 (1)

PRINT# で文字列として送信、INPUT#で文字列として受信

MPC-N816	MPC-816
10　　CNFG#　1 "9600b8pns1NONE" 20　　A1=123 30　　PRINT#　1 A1 "¥r¥n"　← 123<cr><lf> を送信 40　　INPUT#　1 A2$ 50　　A2=VAL(A2$) 60　　PRINT　A2 #RUN 456　　← 456<cr> 受信	10　　CNFG# 4,1,2 20　　A1=123 30　　PRINT# A1 40　　INPUT# A2 50　　PRINT A2 >RUN 456　← 456<cr> 受信 >

- RS-232 (2)

1 キャラクタずつ送信受信

MPC-N816	MPC-816
10　　CNFG#　1 "9600b8pns1NONE"	10　　CNFG# 4,1,2

```
20      PRINT#    1 CHR$(&H00000031) CHR$(&H00000032) CHR$(&H00000033)
30      PRINT#    1 CHR$(&H0000000D) CHR$(&H0000000A)
40      WAIT    LOF(1)>2                    ← バッファに溜まるのを待つ
50      FOR    I=0 TO 2
60       INPUT#    1 CHR_C|1 A$             ← 1キャラずつ入力
70       PRX    ASC(A$)
80      NEXT
#RUN

00000034                                    ← 4 受信
00000035                                    ← 5 受信
00000036                                    ← 6 受信
```

```
20     A1=123
30    PUT# &H31,&H32,&H33
40    PUT# &H0D,&H0A
50    WAIT RS(1)>2
60    FOR I=0 TO 2
70     PRX GET#(0)
80    NEXT I
>RUN
 &H34    ← 4 受信
 &H35    ← 5 受信
 &H36    ← 6 受信
```

- RS-232 (3)

INPUT#にターミネータとタイムアウトを設定して受信

MPC-N816

```
10        CNFG#    1 "9600b8pns1NONE"
20        INPUT#    1 EOL|13 TMOUT|3 A$     ← ターミネタ<cr>, タイムアウト3sec
30        IF    rse_==1 THEN                ← rse_ はタスクローカル予約変数
40         PRINT    "time out"
50         ELSE
60         PRINT    A$
70        END_IF
RUN

 time out                                   ← タイムアウトの場合
RUN

 123                                        ← 123<cr> 受信
```

MPC-816

```
None
```

- パルス発生 (1)

X軸とU軸が交互に動く

MPC-816のパルスコマンドはパルス発生が終了しないと次のステップに進みませんが、MPC-N816のパルスコマンドは実行後すぐに次のステップへ進むので、RR()でパルス発生が終了するのを待ちます。

MPC-N816 + MPG-2541 (or MPG-2314)

```
10        PG    10                          ← MPG宣言
20        ACCEL    X_A|U_A 10000 1000 100   ← 最高速、加減速設定
30        FEED    X_A|U_A 100               ← FEED 1～100(最高速)
40        CLRPOS    X_A|U_A                 ← 現在位置クリア
50        DO
60         MOVS    X_A 10000                ←X軸絶対座標移動
70         WAIT    RR(X_A)==0              ←X軸移動完了待ち
80         MOVS    U_A 10000                ←U軸絶対座標移動
90         WAIT    RR(U_A)==0              ←U軸移動完了待ち
100        CP                               ←現在位置表示
110        RMVS    X_A -10000               ←X軸相対座標移動
120        WAIT    RR(X_A)==0
130        RMVS    U_A -10000               ←U軸相対座標移動
140        WAIT    RR(U_A)==0
150        CP
160       LOOP
RUN

 X=10000 Y=0 U=10000 Z=0
 X=0 Y=0 U=0 Z=0
 X=10000 Y=0 U=10000 Z=0


※次のような RMVS の複数軸指定はできません(MPG-2314も不可)。
  RMVS X_A|U_A 10000
```

MPC-816

```
10   PG 1
20   MODE 5
30   ACCEL 10000,1000,100
40   FEED 0
50   SETPOS 0,0
60   STPZU 0,0,0
70   *LOOP
80   MOVE 10000,0
90   MOVZ 0,10000
100  PRINT P(0)
110  TIME 10
120  RMOV -10000,0
130  RMVZ 0,-10000
140  PRINT P(0)
150  TIME 10
160  GOTO *LOOP
>RUN
 10000 0 0 10000
 0 0 0 0
 10000 0 0 10000
```

- パルス発生 (2)

X軸とU軸同時スタート。MPG-2541は直線補間できません。

MPC-N816 + MPG-2541 (or MPG-2314)

MPC-816

```
10         PG   10
20         ACCEL   X_A|U_A 10000 1000 100
30         FEED   X_A|U_A 100
40         CLRPOS   X_A|U_A
50         DO
60          MOVS   5000 VOID 10000 VOID    ←XU 軸絶対座標移動
70          WAIT   RR(X_A|U_A)==0          ←XU 軸移動完了待ち
80          CP
90          RMVS   -5000 0 -10000 0        ←XU 軸相対座標移動
100         WAIT   RR(X_A|U_A)==0
110         CP
120        LOOP
RUN

 X=5000 Y=0 U=10000 Z=0
 X=0 Y=0 U=0 Z=0
 X=5000 Y=0 U=10000 Z=0
```

None

- パルス発生 (3)

タスク 0 で X 軸、タスク 1 で U 軸を非同期制御。

MPC-N816 + MPG-2541 (or MPG-2314)

MPC-816

```
10         FORK   1 *U_AXIS                ←タスク起動
20         PG   10                         ←タスク 0 の PG 宣言
30         ACCEL   X_A 10000 1000 100
40         FEED   X_A 100
50         CLRPOS   X_A
60         PRINT   "X-->" X(0)
70         DO
80          FOR   sp_=20 TO 100 STEP 20
90           FEED   X_A sp_
100          MOVS   X_A 5000
110          WAIT   RR(X_A)==0
120          PRINT   "X" sp_ X(0)
130          RMVS   X_A -5000
140          WAIT   RR(X_A)==0
150          PRINT   "X" sp_ X(0)
160         NEXT
170        LOOP
180      *U_AXIS
190        PG   10                      ←タスク 1 の PG 宣言(この場合はタスク 0 と同一 PG)
200        ACCEL   U_A 5000 100 100
210        FEED   U_A 100
220        CLRPOS   U_A
230        PRINT   "U-->" U(0)
240        DO
250         FOR   sp_=100 TO 20 STEP -20
260          FEED   U_A sp_
270          MOVS   U_A 2000
280          WAIT   RR(U_A)==0
290          PRINT   "U" sp_ U(0)
300          RMVS   U_A -2000
310          WAIT   RR(U_A)==0
320          PRINT   "U" sp_ U(0)
330         NEXT
340        LOOP
#RUN

 X--> 0
 U--> 0
 U 100 2000
 U 100 0
 U 80 2000
 U 80 0
 X 20 5000
 U 60 2000
 U 60 0
```

None

- パルス発生 (4)

タスク 0 で U 軸制御。パルス発生中に SW が入ったらパルスを停止してプログラム終了。

MPC-N816 + MPG-2541 (or MPG-2314)	MPC-816

```
10        PG   10
20       axis=U_A
30        ACCEL   axis 10000 1000 100
40        CLRPOS   axis
50        DO
60         MOVS   axis 10000                ←絶対座標移動
70         GOSUB   *WAIT_PLS_STOP          ←停止確認サブルーチン
80         _VAR   stp_condition_            ←サブルーチン戻り値=停止状態
90         IF   stp_condition_<>0 THEN      ←SW 停止なら
100         BREAK                           ←DO～LOOP から出る
110        END_IF
120        TIME   500
130        MOVS   axis 0
140        GOSUB   *WAIT_PLS_STOP
150        _VAR   stp_condition_
160        IF   stp_condition_<>0 THEN
170         BREAK
180        END_IF
190        TIME   500
200       LOOP
210       PRINT   "Program has ended"
220       END
230      *WAIT_PLS_STOP
240       WAIT   (RR(axis)==0)|SW(192)==1 ←通常停止 または SW オン
250       IF   SW(192)==1 THEN
260        STOP   STP_I                    ←即停止
270        WAIT   RR(axis)==0              ←停止確認
280        PRINT   "forced stop"
290      stp_condition_=1
300        ELSE
310        PRINT   "nomal stop"
320      stp_condition_=0
330       END_IF
340       RETURN   stp_condition_          ←戻り値
#RUN

 nomal stop
 nomal stop
 forced stop                              ←SW オンで停止
 Program has ended
```

- タスク制御 (1)

タスク 0 はタスク 1 を起動後自己消滅、FTMW にプロンプトが戻ってきてダイレクコマンドが実行できます。
タスク 1 は 10 回オンオフ後自己消滅します。

MPC-N816	MPC-816

```
10        QUIT   1                                  10       FORK 1,*TASK1
20        FORK   1 *TASK1                           20       END
30        END                              ←タスク 0 消滅   30       *TASK1
40       *TASK1                                     40       FOR I=1 TO 10
50        FOR   I=1 TO 10                           50        ON 0
60         ON   0                                   60        TIME 50
70         TIME   500                       ←delay 0.5sec  70        OFF 0
80         OFF   0                                  80        TIME 50
90         TIME   500                               90       NEXT I
100       NEXT   I                                  100      END
110       END                              ←タスク 1 消滅
```

- タスク制御 (2)

タスク 0 はタスク 1 を起動後 SW オン待ち。タスク 1 はオンオフを繰り返します。
タスク 0 は SW がオンしたらタスク 1 を QUIT します。

MPC-N816	MPC-816

```
10        QUIT   1                                  0       FORK 1,*TASK1
```

```
20        FORK   1 *TASK1
30        WAIT   SW(192)==1            ←SW オン待ち
40        QUIT   1                     ←タスク停止
50        WAIT   TASK(1)==255          ←タスク停止確認
60        PRINT   "TASK 1 QUIT" TASK(1)
70        END
80      *TASK1
90        ON   0
100       TIME   500
110       OFF   0
120       TIME   500
130       GOTO  *TASK1
#RUN

 TASK 1 QUIT 255
```

```
20      WAIT SW(0)=1
30      QUIT 1
40      WAIT TASK(1)=7
50      'QUIT 1
60      PRINT STR(-1),TASK(1)
70      END
80      *TASK1
90      ON 0
100     TIME 50
110     OFF 0
120     TIME 50
130     GOTO *TASK1
>RUN
 QUIT 1 7
```

- タスク制御 (3)

タスク 0 はタスク 1 を起動後 SW オン待ち。タスク 1 は X 軸を制御。
タスク 0 は SW がオンしたらタスク 1 停止とパルス停止。
(注意)MPC-N816 と MPC-816 の QUIT と STOP の順番が逆になっています。

MPC-N816 + MPG-2541 (or MPG-2314)

```
10        PG   10
20       axis=X_A
30        QUIT   1
40        FORK   1 *TASK1
50        WAIT   SW(192)==1
60        QUIT   1                     ←タスク停止
70        STOP   axis STP_D            ←パルス減速停止
80        WAIT   RR(axis)==0           ←停止確認
90        PRINT   "TASK 1 QUIT" TASK(1)
100       END
110     *TASK1
120       PG   10
130       ACCEL   axis 10000 1000 100
140       CLRPOS   axis
150     *TASK11
160       MOVS   axis 10000            ←絶対座標移動
170       WAIT   RR(axis)==0
180       TIME   500
190       MOVS   axis 0
200       WAIT   RR(axis)==0
210       TIME   500
220       GOTO  *TASK11
#RUN

 TASK 1 QUIT 255
```

MPC-816

```
10      PG 1
20      FORK 1,*TASK1
30      WAIT SW(0)=1
40      STOP 1
50      WAIT BSY(1)<>0
60      'BSY
70      PRINT STR(-1),BSY(1)
80      QUIT 1
90      WAIT TASK(1)=7
100     'QUIT 1
110     PRINT STR(-1),TASK(1)
120     END
130     *TASK1
140     MODE 5
150     ACCEL 10000,1000,100
160     SETPOS 0,0
170     *TASK11
180     MOVE 10000,0
190     TIME 10
200     MOVE 0,0
210     TIME 10
220     GOTO *TASK11
>RUN
 BSY 15
 QUIT 1 7
```

- タスク制御 (4)

実行しているプログラムを Ctrl+A で停止したとき表示される文番号の後ろに「！」マークが付く場合があります。これはタスクディスパッチの非効率を表すもので、このようなタスクがあると全体的に動作が遅くなります。TIME や SWAP を入れてみてください。

MPC-N816

◆非効率なプログラム

```
10        QUIT   1
20        FORK   1 *TASK1
30       I=0
40        DO
50         IF   SW(192)==1 THEN
60          BREAK
70         END_IF
80        LOOP
90        PRINT   "Task0 has ended"
100       END
110     *TASK1
120       DO
```

MPC-816

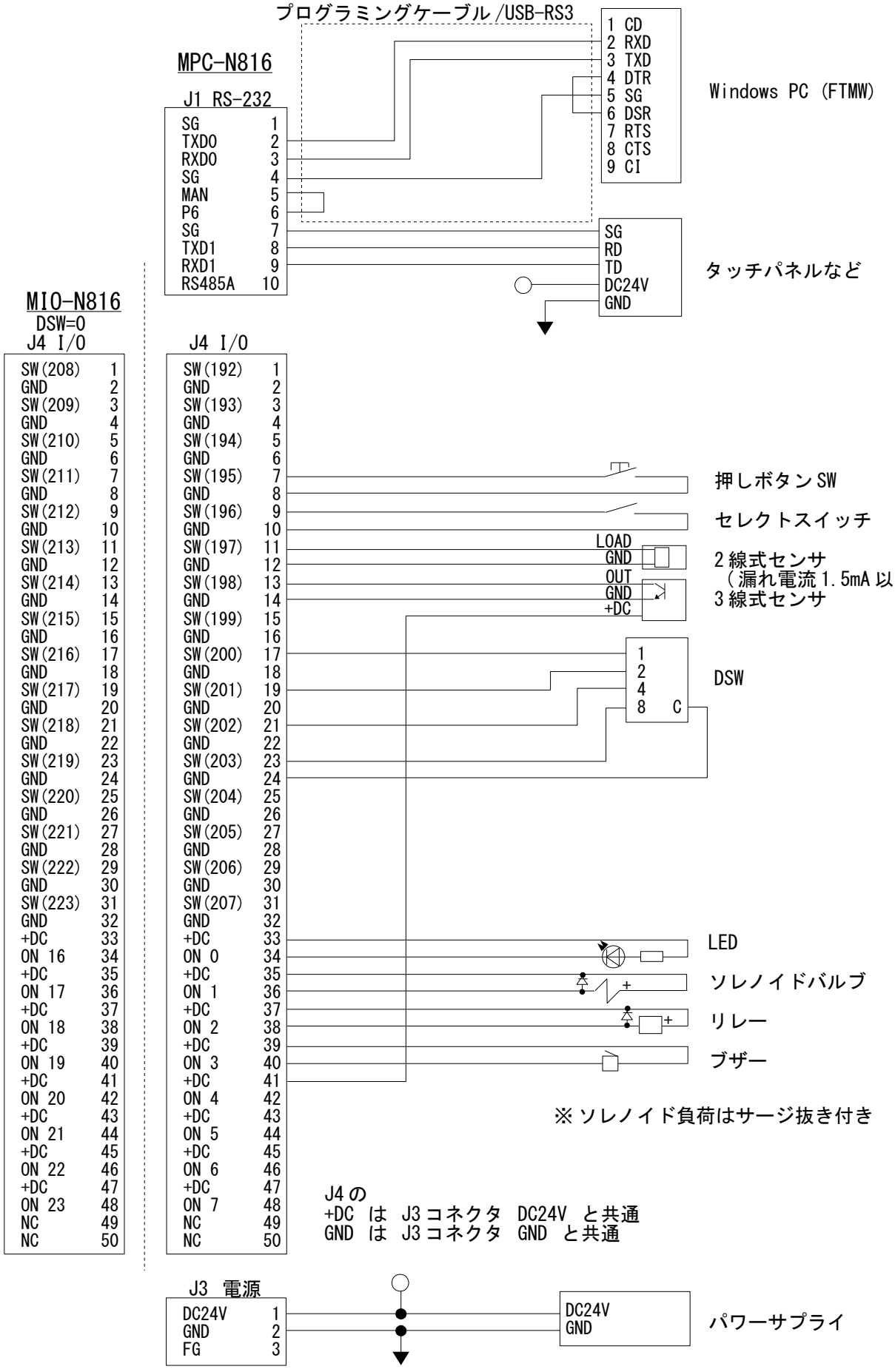
None

```
130        OUT   IN(24) 0
140       LOOP
#RUN
                                          ←Ctrl+A 停止
        *0!   [40]      *1!   [120]       ←タスク番号に「!」 が付く
        The task ! marked is wasting time!! ←日本語 ”!は時間浪費タスクです。”
#
```

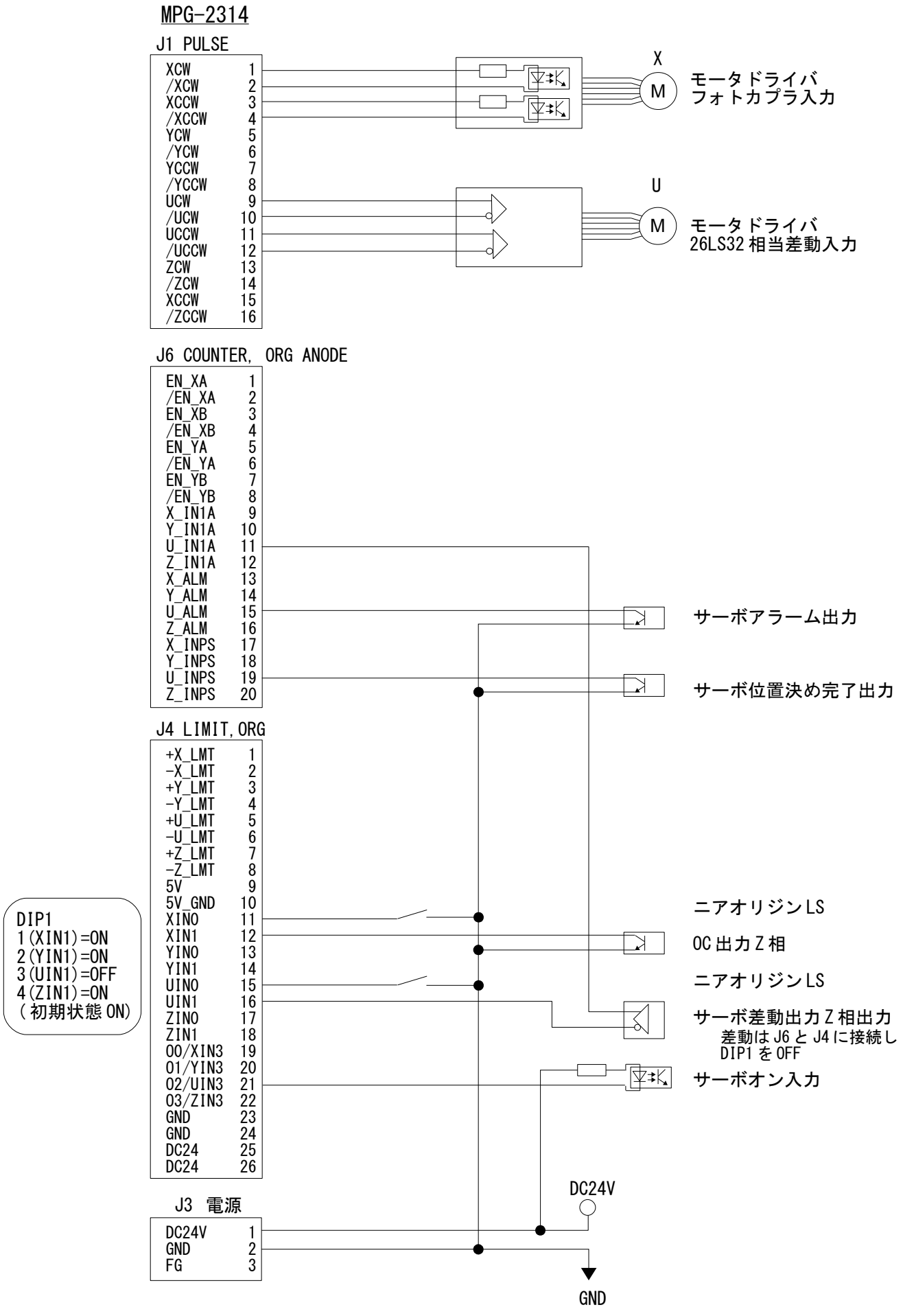
◆↓改善プログラム

```
10        QUIT   1
20        FORK   1 *TASK1
30       I=0
40        DO
50         IF   SW(192)==1 THEN
60          BREAK
70         END_IF
80         SWAP                           ←SWAP を入れてみた
90        LOOP
100       PRINT   "Task0 has ended"
110       END
120     *TASK1
130       DO
140        OUT   IN(24) 0
150        TIME   10                      ← TIME を入れてみた
160       LOOP
#RUN
                                          ←Ctrl+A 停止
        *0    [80]      *1    [150]       ←「!」 が無くなりました
```

MPC-N816 機器接続例



MPG-2314 機器接続例



MPG-2314 I/O とコマンド対応

◆MPG-2314 の I/O チェックは INCHK コマンドです。

```
#PG 0                                          ←PG 指定(DSW の設定によります)
#INCHK                                         ←INCHK コマンドを実行すると一定間隔で入力を読みます
 MPG-2314
 X=+LMT:off-LMT:off ALM:off INP:off IN0:on  IN1:off   ← IN0 がオンしています
 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:on    ← IN1 がオンしています
 Z=+LMT:off-LMT:off ALM:off INP:off IN0:off IN1:off
#                                                    ← 停止は Q または q キー
```

・表示と J4 コネクタ ピン番号([]内)対応

```
 X= +LMT:[J4-1] -LMT:[J4-2] ALM:[J6-13] INP:[J6-17] IN0:[J4-11] IN1:[J4-12]
 Y= +LMT:[J4-3] -LMT:[J4-4] ALM:[J6-14] INP:[J6-18] IN0:[J4-13] IN1:[J4-14]
 U= +LMT:[J4-5] -LMT:[J4-6] ALM:[J6-15] INP:[J6-19] IN0:[J4-15] IN1:[J4-16]
 Z= +LMT:[J4-7] -LMT:[J4-8] ALM:[J6-16] INP:[J6-20] IN0:[J4-17] IN1:[J4-18]
```

◆プログラムで使うコマンド・関数の対応です

J4 PIN	SIGNAL (INCHK の表記)	コマンド・関数
1	+X_LMT (X +LMT)	LMT(X_A,LMTp)
2	-X_LMT (X -LMT)	LMT(X_A,LMTn)
3	+Y_LMT (Y +LMT)	LMT(Y_A,LMTp)
4	-Y_LMT (Y -LMT)	LMT(Y_A,LMTn)
5	+U_LMT (U +LMT)	LMT(U_A,LMTp)
6	-U_LMT (U -LMT)	LMT(U_A,LMTn)
7	+Z_LMT (Z +LMT)	LMT(Z_A,LMTp)
8	-Z_LMT (Z -LMT)	LMT(Z_A,LMTn)
9	5V	
10	5V_GND	
11	XIN0 (X IN0)	HPT(XIN0)
12	XIN1 (X IN1)	HPT(XIN1)
13	YIN0 (Y IN0)	HPT(YIN0)
14	YIN1 (Y IN1)	HPT(YIN1)
15	UIN0 (U IN0)	HPT(UIN0)
16	UIN1 (U IN1)	HPT(UIN1)
17	ZIN0 (Z IN0)	HPT(ZIN0)
18	ZIN1 (Z IN1)	HPT(ZIN1)
19	O0 / XIN3	出力 HOUT &H1 / 入力 HPT(XIN3)
20	O1 / YIN3	出力 HOUT &H2 / 入力 HPT(YIN3)
21	O2 / UIN3	出力 HOUT &H4 / 入力 HPT(UIN3)
22	O3 / ZIN3	出力 HOUT &H8 / 入力 HPT(ZIN3)
23	GND	
24	GND	
25	DC24	
26	DC24	

```
(例)#PG 0
    #PRINT LMT(X_A,LMTp)
     1                          ←Low(GND)状態
    #INSET X_A LMT_OFF          ←High(開放)で有効設定
    #PRINT LMT(X_A,LMTp)
     0                          ←Low(GND)状態
    #PRINT LMT(X_A,LMTp)
     1                          ←High(開放)状態
```

(例) PRINT HPT(XIN0)

(例) PRINT HPT(XIN3)

J6 PIN	SIGNAL (INCHK の表記)	コマンド・関数
13	X_ALM (X ALM)	LMT(X_A,ALM)
14	Y_ALM (Y ALM)	LMT(Y_A,ALM)
15	U_ALM (U ALM)	LMT(U_A,ALM)
16	Z_ALM (Z ALM)	LMT(Z_A,ALM)
17	X_INPS (X INP)	
18	Y_INPS (Y INP)	
19	U_INPS (U INP)	
20	Z_INPS (Z INP)	

```
(例)#PG 0
    #INSET ALL_A ALM_ON   ←Low(GND)でオンの設定
    #PRINT LMT(X_A,ALM)   ←表示
     1                    ←Low(GND)状態
```

◆入力機能の設定例。入力機能の設定は INSET コマンドです。

```
INSET X_A ALM_ON|LMT_ON      X 軸 ALM,LMT は Low(GND)でアクティブ
INSET Y_A ALM_OFF            Y 軸 ALM は High(開放)でアクティブ
INSET U_A LMT_OFF            U 軸 LMT は High(開放)でアクティブ
INSET ALL_A INP_ON           全軸 INPOS は Low(GND) でアクティブ
INSET ALL_A INP_OFF          全軸 INPOS は High(開放) でアクティブ
```

ALM,INP は INSET 設定することで有効になります。LMT は常時有効で無効にすることはできません。

パルス発生の確認

配線チェック時のパルス発生の確認はティーチングモードが簡単です。(MPG-2314､MPG-2541 共通)

```
#PG 0                                          /* PG 指定
#ACCEL ALL_A 10000                             /* 速度設定
#CLRPOS                                        /* 現在位置クリア
#T                                             /* ティーチングモード
PG=[0]@00  X=800 Y=-600 U=1000 Z=0 dx=200 dy=200 du=200 dz=200
              ↑現在座標値                      ↑移動量
               'x,y,u,z' key で CW               '0,1,2,3' key で指定
               'X,Y,U,Z' key で CCW
```

MPG ボード上の LED1(XCW)～LED8(ZCCW)でパルス出力を確認できます(パルス数・スピードによっては LED の点灯が分かりにくい場合があります)。

'Q,q'でティーチングモードから抜けます。

移動量の変更は SET コマンドです。

```
#SET 0 1 1 1 1                  /* '0' key の設定
#SET 1 10 10 10 10              /* '1' key の設定
#SET 2 100 100 100 100          /* '2' key の設定
#SET 3 1000 1000 1000 1000      /* '3' key の設定
```

MPC-N816 の電源を切ると SET 内容は初期状態に戻ります。

MPG-2314 原点復帰例

◆PRG1
次の (1)ニアオリジン検出と (2)原点検出を連続して行います。

(1) CCW 方向へニアオリジン(XIN0)がオンするまで移動→XIN0 がオンすると減速停止
減速域は ACCEL の leng

XIN0 オンまで <--- CCW 方向　最高スピード=1000pps、 加減速有り、加減速域=500pulse

(2) CW 方向へオリジン(XIN1=Z 相)がオンするまで移動→XIN1 がオンすると即停止
CW 方向の移動速度は ACCEL の lo_pps となります。

スピード=100pps、加減速無し CW 方向 ---> XIN1 オンまで

入力：XIN0、XIN1
論理：どちらもノーマリオープン(メカが叩いてオン)

```
PG 0                                    /* MPG-2314 は PG 0～
ACCEL X_A 1000 500 100                  /* 原点復帰速度 IN0 オンまで 1000、IN1 オンまで 100pps
FEED 100
SHOM X_A IN0_ON|IN1_ON|CW               /* 停止条件 IN0,IN1 オン。IN0 検出後 CW 方向に Z 相検出
HOME NEG_L 0 0 0                        /* 原点復帰 IN0 まで CCW
CP
```

◆PRG2
通常、オリジン(IN1)はニアオリジン(IN0)との組み合わせで使用しますが、下記はオリジンだけで原点復帰する例です。この場合の移動速度は ACCEL の初速度(この例では 1000pps)で即停止します。

入力：XIN1、(XIN0 未接続)
論理：ノーマリオープン(メカが叩いてオン)

```
PG 0
ACCEL X_A 10000 5000 1000               /* 初速度 1000pps
FEED 100                                /* XIN1 オン/オフ確認
IF HPT(XIN1)==1 THEN
  RMVS X_A 1000                         /* 退避移動
  WAIT RR(X_A)==0
END_IF
SHOM X_A IN0_OFF|IN1_ON|CCW             /* 停止条件
HOME X_A NEG_L                          /* 原点復帰 CCW
CP
```

◆PRG3
ニアオリジン(IN0)停止の場合、ACCEL コマンドの加減速設定が有効なので、IN0 がオンしてから減速領域分移動します。次は加減速無しの実行例です。

入力：XIN0、(XIN1 未接続)
論理：ノーマリオープン(メカが叩いてオン)

```
PG 0
FEED 100
IF HPT(XIN0)==1 THEN                    /* XIN0 オン/オフ確認
  ACCEL X_A 10000 5000 1000             /* 退避移動の速度
  RMVS X_A 1000                         /* 退避移動
  WAIT RR(X_A)==0
END_IF
ACCEL X_A 1000 500 1000                 /* 原点復帰の速度。最高速=初速度
SHOM X_A IN0_ON                         /* 停止条件
HOME X_A NEG_L                          /* 原点復帰 CCW
CP
```

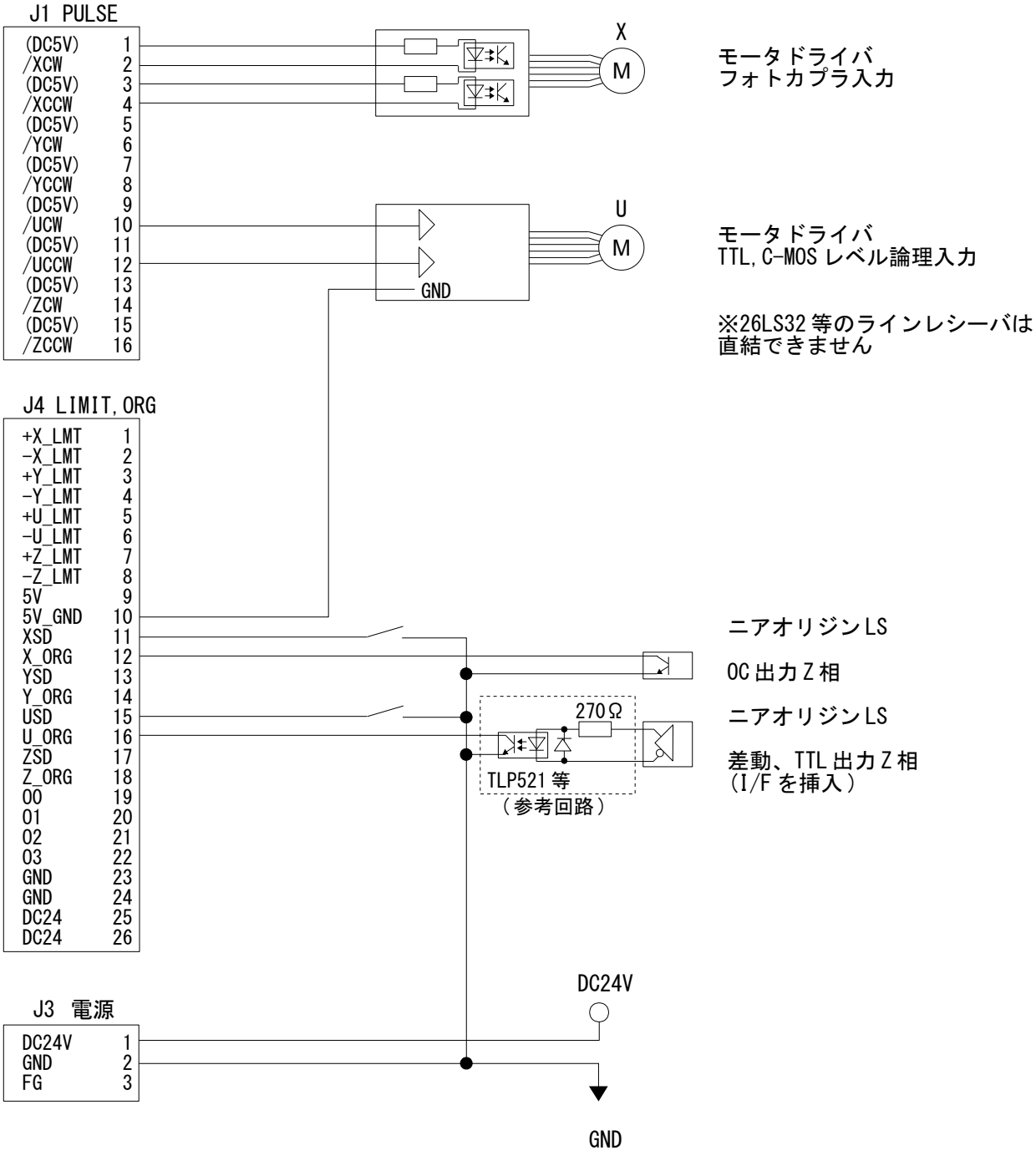
◆PRG4
ニアオリジンを使ったＸ軸Ｙ軸同時原点復帰例。最高速と初速度を同じにして加減速を無くしています。

入力：XIN0、YIN0
論理：ノーマリオープン(メカが叩いてオン)

```
PG 0
ACCEL X_A|Y_A 10000 500 100             /* 原点復帰スピード
FEED 100

IF HPT(XIN0)<>0 THEN                    /* X 軸 IN0 がオンなら退避移動
  RMVS X_A 10000
END_IF
IF HPT(YIN0)<>0 THEN                    /* Y 軸 IN0 がオンなら退避移動
  RMVS Y_A 10000
END_IF
WAIT RR(ALL_A)==0

ACCEL X_A|Y_A 1000 500 1000             /* 原点復帰速度 1000pps
SHOM X_A|Y_A IN0_ON                     /* 停止条件 X,Y 軸 IN0 がオン
HOME NEG_L NEG_L 0 0                    /* 原点復帰
CP
```

MPG-2541 機器接続例

MPG-2541



※INPOS 入力はありません。必要に応じて一般の入力へ接続してください。

MPG-2541 I/O とコマンド対応

◆MPG-2541 の I/O チェックは MPG-2314 と同じ INCHK コマンドです。

```
#PG 10                                  ← PG 指定 (DSW の設定によります)
#INCHK                                  ← INCHK コマンドを実行すると一定間隔で入力を読みます
 MPG-2541
 X= +EL:1 -EL:0  SD:0 ORG:0 OTS:0  ← +EL がオンしています
 Y= +EL:0 -EL:0  SD:0 ORG:0 OTS:0
 U= +EL:0 -EL:0  SD:1 ORG:0 OTS:0  ← SD がオンしています
 Z= +EL:0 -EL:0  SD:0 ORG:0 OTS:0
#                                       ← 停止は Q または q キー
```

・表示と J4 コネクタ ピン番号([]内)対応

```
 X= +EL:[1]  -EL:[2]  SD:[11]  ORG:[12]  OTS:[19(OUT)]
 Y= +EL:[3]  -EL:[4]  SD:[13]  ORG:[14]  OTS:[20(OUT)]
 U= +EL:[5]  -EL:[6]  SD:[15]  ORG:[16]  OTS:[21(OUT)]
 Z= +EL:[7]  -EL:[8]  SD:[17]  ORG:[18]  OTS:[22(OUT)]
```

※OTS は出力の状態

◆プログラムで使うコマンド・関数の対応です

<table>
<tr><th>J4 PIN</th><th>SIGNAL (INCHK の表記)</th><th colspan="2">コマンド・関数</th></tr>
<tr><td>1</td><td>+X_LMT (X +EL)</td><td colspan="2" rowspan="8">LMT(0)
パラレル入力
-Z_LMT 上位～+X_LMT 下位
bit 入力はありません
(例)
#PRX LMT(0)
00000001 ← +X_LMT オン</td></tr>
<tr><td>2</td><td>-X_LMT (X -EL)</td></tr>
<tr><td>3</td><td>+Y_LM (Y +EL)</td></tr>
<tr><td>4</td><td>-Y_LMT (Y -EL)</td></tr>
<tr><td>5</td><td>+U_LMT (U +EL)</td></tr>
<tr><td>6</td><td>-U_LMT (U -EL)</td></tr>
<tr><td>7</td><td>+Z_LMT (Z +EL)</td></tr>
<tr><td>8</td><td>-Z_LMT (Z -EL)</td></tr>
<tr><td>9</td><td>5V</td><td colspan="2"></td></tr>
<tr><td>10</td><td>5V_GND</td><td colspan="2"></td></tr>
<tr><td>11</td><td>XSD (X SD)</td><td>bit 入力 HPT(1)</td><td rowspan="8">HPT(0)
パラレル入力
Z_ORG 上位～XSD 下位
(例)
#PRX HPT(0)
00000010 ← USD オン</td></tr>
<tr><td>12</td><td>X_ORG (X ORG)</td><td>bit 入力 HPT(2)</td></tr>
<tr><td>13</td><td>YSD (Y SD)</td><td>bit 入力 HPT(4)</td></tr>
<tr><td>14</td><td>Y_ORG (Y ORG)</td><td>bit 入力 HPT(8)</td></tr>
<tr><td>15</td><td>USD (U SD)</td><td>bit 入力 HPT(16)</td></tr>
<tr><td>16</td><td>U_ORG (U ORG)</td><td>bit 入力 HPT(32)</td></tr>
<tr><td>17</td><td>ZSD (Z SD)</td><td>bit 入力 HPT(64)</td></tr>
<tr><td>18</td><td>Z_ORG (Z ORG)</td><td>bit 入力 HPT(128)</td></tr>
<tr><td>19</td><td>O0 (X OTS)</td><td colspan="2">bit 出力 H_ON 0, H_OFF 0</td></tr>
<tr><td>20</td><td>O1 (Y OTS)</td><td colspan="2">bit 出力 H_ON 1, H_OFF 1</td></tr>
<tr><td>21</td><td>O2 (U OTS)</td><td colspan="2">bit 出力 H_ON 2, H_OFF 2</td></tr>
<tr><td>22</td><td>O3 (Z OTS)</td><td colspan="2">bit 出力 H_ON 3, H_OFF 3</td></tr>
<tr><td>23</td><td>GND</td><td colspan="2"></td></tr>
<tr><td>24</td><td>GND</td><td colspan="2"></td></tr>
<tr><td>25</td><td>DC24</td><td colspan="2"></td></tr>
<tr><td>26</td><td>DC24</td><td colspan="2"></td></tr>
</table>

MPG-2541 原点復帰例

◆PRG1
XSD(ニアオリジン)を検出したら減速し、X_ORG(オリジン)検出で停止します。

入力：XSD、X_ORG
論理：どちらもノーマリオープン(メカが叩いてオン)

```
  PG 10                 /* MPG-2541 はアドレス 10～
  ACCEL X_A 1000        /* 原点復帰の速度 XSD まで 1000pps、XSD を検出したら 50pps ※
  FEED X_A 100          /* 必ず FEED を設定
  SHOM 0                /* SHOM の設定

  IF HPT(1)==1 THEN     /* XSD が入っていたら
    RMVS X_A 1000       /* 退避移動
    WAIT RR(X_A)==0
  END_IF

  HOME X_A NEG_L        /* XSD オン:減速→XORG オン:停止
  CLRPOS X_A            /* Clear Position


/* ※ACCEL コマンドの最低速を省略すると 最高速の 1/20 になります。
/*   ACCEL コマンドをパラメータ無しで実行すると現在値を表示します。
```

◆PRG2

原点センサは１つで X_ORG に接続した場合。

入力： X_ORG
論理：ノーマリオープン(メカが叩いてオン)

```
  PG 10                 /* MPG-2541 はアドレス 10～
  ACCEL X_A 1000        /* 原点復帰の速度
  FEED X_A 100          /* 必ず FEED を設定
  SHOM 0                /* SHOM の設定

  IF HPT(2)==1 THEN     /* XORG が入っていたら
    RMVS X_A 1000       /* 退避移動
    WAIT RR(X_A)==0
  END_IF

  HOME X_A NEG_L        /* XORG オン:停止
  CLRPOS X_A            /* Clear Position
```

◆PRG3
原点センサは１つで XSD に接続した場合。HOME コマンドは使わず STOP コマンドを使います。

入力： XSD
論理：ノーマリオープン(メカが叩いてオン)

```
  PG 10                 /* MPG-2541 はアドレス 10～
  ACCEL X_A 1000        /* 原点復帰の速度
  FEED X_A 100          /* 必ず FEED を設定
  SHOM 0                /* SHOM の設定解除

  IF HPT(1)==1 THEN     /* XSD が入っていたら
    RMVS X_A 1000       /* 退避移動
    WAIT RR(X_A)==0
  END_IF

  RMVS X_A -30000       /* CCW 方向に移動する
  WAIT HPT(1)==1        /* XSD がオン待ち
  STOP X_A STP_I        /* 即停止
  WAIT RR(X_A)==0
  CLRPOS X_A            /* 座標クリア
  CP                    /* 現在座標表示
```

◆PRG4
ニアオリジンを用いた X 軸 U 軸同時原点復帰例です。

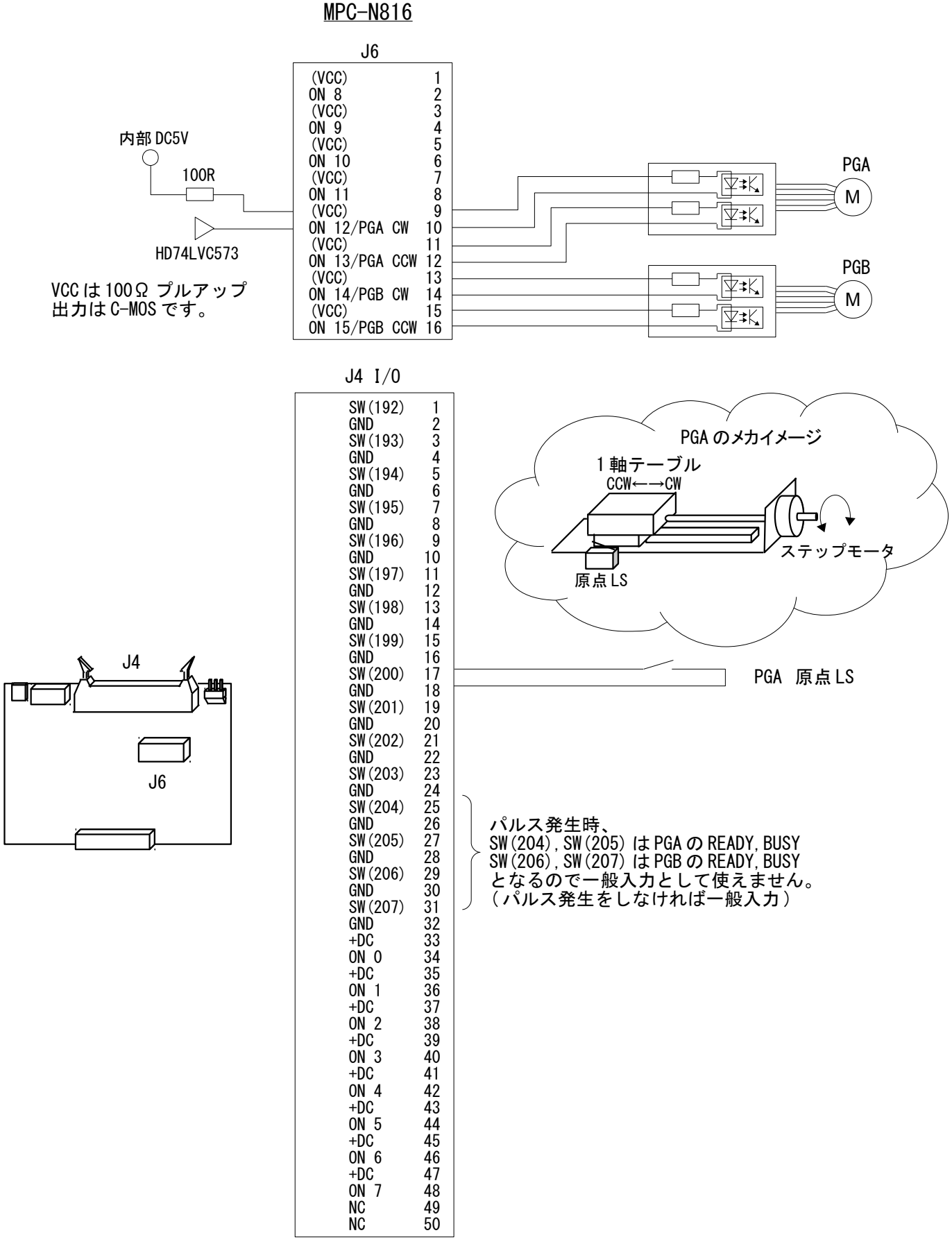
XUの2軸同時原点復帰
入力： XSD,USD
論理：ノーマリオープン(メカが叩いてオン)

```
  PG 10                   /* MPG-2541はアドレス10～
  axis=X_A|U_A
  ACCEL axis 1000         /* 原点復帰の速度
  FEED axis 100           /* 必ずFEEDを設定
  SHOM 0                  /* SHOMの設定解除

  IF HPT(1)==1 THEN       /* XSDが入っていたら
    RMVS X_A 1000         /* X 退避移動
  END_IF
  IF HPT(&H10)==1 THEN    /* USDが入っていたら
    RMVS U_A 1000         /* U 退避移動
  END_IF
  WAIT RR(axis)==0        /* 退避移動完了待ち

  RMVS NEG_L VOID NEG_L VOID /* CCW方向に移動する
  DO
    IF (HPT(1)==1)&(RR(X_A)<>0) THEN  /* XSDがオン AND Xパルス出力中
      STOP X_A STP_I      /* 即停止
    END_IF
    IF (HPT(&H10)==1)&(RR(U_A)<>0) THEN  /* USDがオン AND Uパルス出力中
      STOP U_A STP_I      /* 即停止
    END_IF
    IF RR(axis)==0 THEN
      BREAK
    END_IF
    SWAP
  LOOP
  CLRPOS axis             /* 座標クリア
  CP                      /* 現在座標表示
```

MPC-N816 J6 コネクタ 簡易パルス発生例

MPC-N816 の J6 コネクタからも 2 軸の簡易的なパルス発生が行えます。
パルス出力は MIF-816 J5 相当の C-MOS 出力です。
制御は MPG-2314、MPG-2541 のパルス発生とは区別され PGA、PGB コマンドで行います。
最高速は約 12Kpps です。補間はできません。マルチタスクで動作します。

◆接続例



◆MPC プログラム例

PGA は 原点復帰後、 CW 方向相対移動 10 回(スピード変化)→CCW 方向絶対座標移動 繰り返し
PGB は CW 方向相対移動→CCW 方向相対移動 繰り返し

```
'****************************************
  QUIT 1 2
  FORK 1 *PGA
  FORK 2 *PGB
  END

'****************************************
*PGA
  PGA "D" 0                   /* "D": パルス出力方式選択  0=CW/CCW(default), 1=PULSE/DIR
  PGA "A" 10000               /* "A": 加減速テーブル作成 500~12000pps 加速距離は最高速の1/10。
                              /*      "A"は演算に少し時間がかかります。
  WAIT SW(204)==1             /* "SW(204)": PGA ready。ここでは、加減速テーブル作成完了待ち

  GOSUB *PGA_HOME

  DO
    FOR I=10 TO 1 STEP -1
      PGA "F" I               /* "F": スピード設定 10(max)~0(min)。
      PGA "R" 1000            /* "R": 相対座標移動
      WAIT SW(204)==1         /* 移動完了待ち
      PGA "C"                 /* "C": 現在位置取得(ちょっとだけ時間がかかります)
      PRINT "PGA pos=" V_PGA  /* "V_PGA": 上記"C"実行後は現在位置が入ります。
    NEXT I

    PGA "F" 10                /* max スピード
    PGA "M" 0                 /* "M": 絶対座標移動。ここでは原点に戻ります。
    WAIT SW(204)==1           /* 移動完了待ち
    PGA "C"
    PRINT "PGA pos=" V_PGA
  LOOP

*PGA_HOME                     /* PGA 原点復帰
  IF SW(200)==1 THEN          /* 原点 LS が入っていたら退避移動
    PGA "F" 5                 /* スピード
    PGA "R" 1000              /* 相対座標移動
    WAIT SW(204)==1           /* 移動完了待ち
  END_IF
  PGA "G" -1000               /* "G": 定速パルス発生。CCW 1000pps
  WAIT SW(200)==1             /* 原点 LS オン待ち
  OFF PGA                     /* PGA パルス発生停止
  TIME 50                     /* 停止後、遅延時間を入れてください
  PGA "H" 0                   /* "H": 現在位置設定。この場合、現在位置を 0 とします。
  PGA "V"                     /* "V": PGA ファームウェアバージョン取得
  PRINT "PGA version=" V_PGA  /* "V_PGA": 上記"V"実行後はバージョンが入ります。
  PGA "C"
  PRINT "PGA pos=" V_PGA
  RETURN

'****************************************
*PGB
  PGB "D" 0                   /* "D": パルス出力方式選択  0=CW/CCW(default), 1=PULSE/DIR
  PGB "A" 10000               /* "A": 加減速テーブル作成 500~12000pps 加速距離は最高速の1/10。
                              /*      "A"は演算に少し時間がかかります。
  WAIT SW(206)==1             /* "SW(206)": PGB ready。ここでは、加減速テーブル作成完了待ち
  PGB "F" 10                  /* max スピード

  DO
    PGB "R" 1000              /* "R": 相対座標移動 CW 1000
    WAIT SW(206)==1           /* 移動完了待ち
    TIME 500
    PGB "R" -1000             /* 相対座標移動 CCW 1000
    WAIT SW(206)==1           /* 移動完了待ち
    TIME 500
  LOOP
```

・実行結果
```
RUN

# PGA version= 100215
 PGA pos= 0
 PGA pos= 1000
 PGA pos= 2000
 PGA pos= 3000
 PGA pos= 4000
 PGA pos= 5000
 PGA pos= 6000
 PGA pos= 7000
 PGA pos= 8000
 PGA pos= 9000
 PGA pos= 10000
 PGA pos= 0
 PGA pos= 1000
 PGA pos= 2000
```

--- End Of Document ---