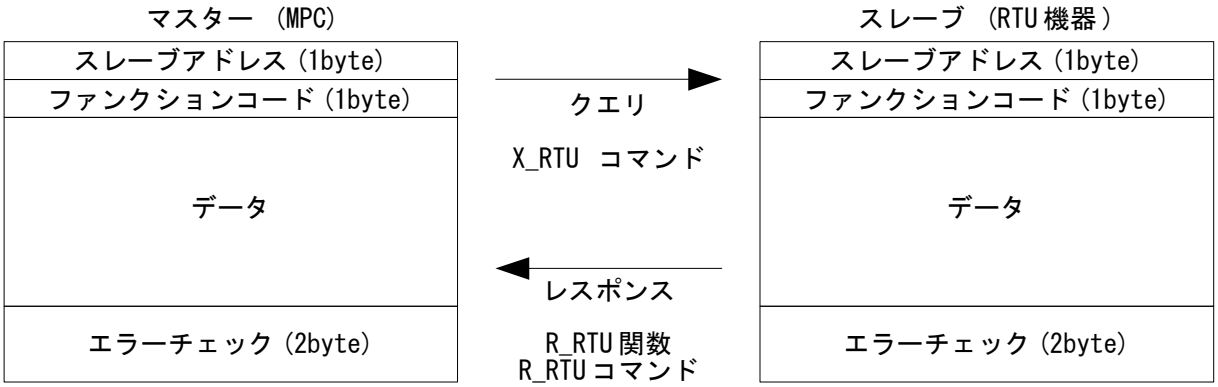
サポート MPC Ver BL/I 1.14_48 2016/01/22 以降

Modbus-RTU プロトコルのイメージ



スレーブアドレスは機器で異なる固有の番号、エラーチェックは CRC-16 で計算します。

クエリ送信

X_RTU コマンドでクエリーを送信します。CRC は自動計算して付加されます。

a) データを１つずつ指定する方法。データ数が少ない場合に向いています。
```
X_RTU SlaveAddress FunctionCode Data0 Data1 .. DataN
```

b) データを配列に入れて指定する方法。データ数が多い場合に向いています。
```
X_RTU SlaveAddress FunctionCode Format QueryArray
```

例)

クエリ

スレーブアドレス 1 (1byte)	ファンクションコード 3 (1byte)	Data0(2byte) 1234	Data1(1byte) 5	Data2(4byte) 123456789

a) の例

```
Slave=1
Code=3
Data0=1234
Data1=5
Data2=123456789
X_RTU Slave Code Data0~Wrd Data1~Byt Data2
```

各データは、1byte:~Byt, 2byte:~Wrd, 4byte:指定無 でキャストします。
変数・数値とちらでも可。 X_RTU 1 3 1234~Wrd 5~Byt 123456789

b) の例

```
DIM QUERY(3)
Slave=1
Code=3
QUERY(0)=1234
QUERY(1)=5
QUERY(2)=123456789
X_RTU Slave Code "WBL" QUERY(0)
```

データが Word,Byte,Long の順なので Format は"WBL"、配列の先頭からなので QUERY(0) を指定。

レスポンス受信

R_RTU 関数で受信後、R_RTU コマンドまたは R_RTU(s,i)でデータをセパレートします。

a) データを１つずつ変数に読み込む方法。データ数が少ない場合の一括読込に向いています。
```
i=R_RTU(bytecount)
R_RTU SlaveAddress FunctionCode Res0 Res1 .. ResN
```

b) データを配列に読み込む方法。データ数が多い場合の一括読込に向いています。
```
i=R_RTU(bytecount)
R_RTU SlaveAddress FunctionCode Format ResArray
```

c) データの指定位置から読み込む方法。特定のデータ読込に向いています。
```
i=R_RTU(bytecount)
Res=R_RTU(size,index)
```

R_RTU(n)には受信バイト数を指定(CRCを除く)。

R_RTU(n)の戻り値。戻りに応じて処理を行ってください。
```
 1: 正常受信
 0: CRCエラー
-1: タイムアウト
```

例)

レスポンス

スレーブアドレス 1 (1byte)	ファンクションコード 3 (1byte)	Res0(2byte) 1234	Res1(1byte) 5	Res2(4byte) 123456789

a) の例

```
i=R_RTU(9)
R_RTU Slave Code Res0~Wrd Res1~Byt Res2
```

各変数は、 1byte:~Byt, 2byte:~Wrd, 4byte:指定無 でキャストします。

b) の例

```
DIM RES(3)
i=R_RTU(9)
R_RTU Slave Code "WBL" RES(0)
```

レスポンスが Word,Byte,Long の順なので Format は"WBL"、RES(0) を指定して配列の先頭から読み込む。この場、RES(0)←Res0,RES(1)←Res1,RES(2)←Res2 となります。

C) の例
```
i=R_RTU(9)
value=R_RTU(Wrd,3)
```

先頭より 3 バイト目から Word 読込 ∴ value=Res0 。

MPC 通信ポートの初期化

MODBUS コマンドで RS-485 ポートを Modbus RTU 通信に割り当てます。(CNFG#は不可)

【書式】
```
MODBUS CH [TASK#] [CRC16 マスク &H 初期値+多項式] ["通信設定文字列"]
```

"通信設定文字列"を指定すると初期化も行われます。仕様は、CNFG#コマンドと共通です。
TASK# は、MODBUS コマンドが実行されるタスク以外で MODBUS ポートを使用するタスクを指定。
CRC16 マスクは、初期値&HFFFF,多項式&HA001 を既定としていますが、変更する場合はヘキサ表現で、初期値を上位、多項式を下位に与えます。
ポートの指定は、起動されたタスクで"MODBUS 3"などのようにすることもできます。

例)
```
MODBUS 2 1 "38400b8pns1NONE"        /* 通信 CH=2, コマンド実行タスク=1
```

オリエンタルモーター ARD-KD

　このサンプルは、２バイトまたは４バイト単位でレジスタを読み書きするサブルーチンを組み合わせてモータ回転(移動)やパラメータの設定などを行います。

保持レジスタから２バイト読み出すサブルーチン

```
'=====================================
' ARD 2バイト読み出し
'=====================================
*ARD_READ_REG_W
_VAR  Ard_RegAdd

/* 保持レジスタの読み出し(2バイト長)
/* Query Example
/* 0203007F0001xxxx
/* 02 スレーブアドレス
/*   03 ファンクションコード
/*     007F 読み出しの起点となるレジスタアドレス (Wrd)
/*         0001 起点のレジスタアドレスから読み出すレジスタの数 (Wrd)
/*             xxxx  CRC
/*
/* Responce Example
/* 0203021234xxxx
/* 02 スレーブアドレス                                          ┐
/*   03 ファンクションコード                                    │ 5バイト
/*     02 データバイト数 (Byt)                                  │
/*       1234 レジスタアドレス 007F の読み出し値 (Wrd)          ┘
/*           xxxx CRC

  DO
    TIME 5 : TMOUT 2000                          /* Silent Interval : Time Out
    X_RTU 2 &H03 Ard_RegAdd~Wrd 1~Wrd
    r_res=R_RTU(5)
    IF r_res==1 THEN
      BREAK
    ELSE
     GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN
```

- コール例。ドライバ出力指令 007Fh を指定してドライバの出力信号を読み出し、RAEDY ビットをチェックします。

```
'=====================================
' ARD レディー待ち
'=====================================
*ARD_WAIT_READY

  prc$="ARD_WAIT_READY" : PR prc$

  DO
    GOSUB *ARD_READ_REG_W &H7F
    IF R_RTU(Byt,5)&&H20<>0 THEN
      BREAK
    END_IF
  LOOP
  RETURN
```

保持レジスタから 4 バイト読み出すサブルーチン

```
'=====================================
' ARD 4バイト読み出し
'=====================================
*ARD_READ_REG_L
_VAR   Ard_RegAdd

/* 保持レジスタの読み出し(4バイト長)
/* Query Example
/* 020300CC0002xxxx
/* 02 スレーブアドレス
/*   03 ファンクションコード
/*     00CC 読み出しの起点となるレジスタアドレス (Wrd)
/*         0002 起点のレジスタアドレスから読み出すレジスタの数 (Wrd)
/*             xxxx  CRC
/*
/* Responce Example
/* 02030412345678xxxx
/* 02 スレーブアドレス                                                    ┐
/*   03 ファンクションコード                                              ├ 7バイト
/*     04 データバイト数 (Byt)                                            ┘
/*       12345678 レジスタアドレス 00CC, 00CD の読み出し値 (Lng)
/*               xxxx CRC

  DO
    TIME 5 : TMOUT 2000                  /* Silent Interval : Time Out
    X_RTU  2 &H03 Ard_RegAdd~Wrd 2~Wrd
    r_res=R_RTU(7)                       /* 受信バイト数を指定する(CRCは含まない)
    IF r_res==1 THEN
      BREAK
    ELSE
     GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN
```

- コール例。レジスタアドレス 00CCh を指定してフィードバック位置を取得します。

```
'=====================================
' ARD 現在位置取得
'=====================================
*ARD_CURRENT_POS

  prc$="ARD_CURRENT_POS" : PR prc$

  GOSUB   *ARD_READ_REG_L &HCC
    Ard_CurPos=R_RTU(Lng,4)                    /* 受信データの4バイト目から4バイト読込
    PR "ARD Current Position=" Ard_CurPos
  RETURN
```

保持レジスタへ 2 バイト書き込むサブルーチン

```
'====================================
' ARD 2バイト書き込み
'====================================
*ARD_WRITE_REG_W
_VAR  Ard_RegAdd Ard_WriteData

/* 保持レジスタへの書き込み(2バイト長)
/* Query Example
/* 0206024B0050xxxx
/* 02 スレーブアドレス
/*   06 ファンクションコード
/*     024B 書き込みを行うレジスタアドレス (Wrd)
/*         0050 レジスタに書き込む値 (Wrd)
/*             xxxx CRC
/*
/* Responce Example
/* 0206024B0050xxxx
/* 02 スレーブアドレス              ┐
/*   06 ファンクションコード        | 6バイト
/*     024B レジスタアドレス (Wrd)  |
/*         0050 ライト値 (Wrd)      ┘
/*             xxxx CRC

  DO
    TIME 5 : TMOUT 2000                          /* Silent Interval : Time Out
    X_RTU 2 &H06 Ard_RegAdd~Wrd Ard_WriteData~Wrd
    r_res=R_RTU(6)
    IF r_res==1 THEN
      BREAK
    ELSE
     GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN
```

- コール例。ドライバ入力指令 007Dh に点番号を指定し、設定した点に移動します。

```
'====================================
' ARD 点移動
'====================================
*ARD_POINT_MOVE
_VAR Ard_PointNum

  prc$="ARD_POINT_MOVE" : PR prc$ Ard_PointNum

  GOSUB *ARD_WRITE_REG_W &H007D &H8|Ard_PointNum   /* ドライバ入力指令 Start|(M2,M1,M0)
  GOSUB *ARD_WRITE_REG_W &H007D &H0
  GOSUB *ARD_WAIT_READY
  GOSUB *ARD_CURRENT_POS

  RETURN
```

保持レジスタへ 4 バイト書き込むサブルーチン

```
'=====================================
' ARD 4バイト書き込み
'=====================================
*ARD_WRITE_REG_L
_VAR  Ard_RegAdd Ard_WriteData

/* 複数の保持レジスタへの書き込み(4バイト長)
/* Query Example
/* 021002C2000204000001F4xxxx
/* 02 スレーブアドレス
/*   10 ファンクションコード
/*     02C2 書き込みの起点となるレジスタアドレス (Wrd)
/*         0002 起点のレジスタアドレスから書き込むレジスタの数 (Wrd)
/*             04 バイト数 (Byt)
/*               000001F4 レジスタアドレス 02C2h, 02C3h の書き込み値 (Lng)
/*                     xxxx  CRC
/*
/* Responce Example
/* 021002C20002xxxx
/* 02 スレーブアドレス              ┐
/*   10 ファンクションコード        | 6バイト
/*     02C2 レジスタアドレス (Wrd)  |
/*         0002 レジスタの数 (Wrd)  ┘
/*             xxxx CRC

  DO
    TIME 5 : TMOUT 2000                  /* Silent Interval : Time Out
    X_RTU 2 &H10 Ard_RegAdd~Wrd 2~Wrd 4~Byt Ard_WriteData   /* クエリー送信
    r_res=R_RTU(6)
    IF r_res==1 THEN
      BREAK
    ELSE
     GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN
```

- コール例。運転データレジスタにデータを書き込みます。

```
'=====================================
' ARD 点の設定
'=====================================
*ARD_POINT_SET

  prc$="ARD_POINT_SET" : PR prc$

  FOR I=0 TO 10 STEP 2
    GOSUB *ARD_WRITE_REG_L &H0400+I I*250       /* 位置
    GOSUB *ARD_WRITE_REG_L &H0480+I 5000        /* 運転速度
    GOSUB *ARD_WRITE_REG_L &H0500+I 1           /* 0:インクリメンタル, 1:アブソリュート
    GOSUB *ARD_WRITE_REG_L &H0580+I 0           /* 0:単軸, 1:連結
    GOSUB *ARD_WRITE_REG_L &H0600+I 1000        /* 加速
    GOSUB *ARD_WRITE_REG_L &H0680+I 1000        /* 減速
  NEXT
  RETURN
```

その他の機能

- 指定量の Jog 移動(相対移動)

```
'=====================================
' ARD Jog移動
'=====================================
*ARD_JOG
_VAR Adr_JogDistance

  prc$="ARD_JOG" : PR prc$ Adr_JogDistance

  GOSUB *ARD_WRITE_REG_L &H0288 1000             /* JOG 加減速
  GOSUB *ARD_WRITE_REG_L &H028A 100              /* JOG 起動速度
  GOSUB *ARD_WRITE_REG_L &H0286 2000             /* JOG 運転速度Hz
  GOSUB *ARD_WRITE_REG_L &H1048 Adr_JogDistance /* JOG 移動量
  GOSUB *ARD_WRITE_REG_W &H007D &H2000           /* ドライバ入力指令 -JOG
  GOSUB *ARD_WRITE_REG_W &H007D &H0
  GOSUB *ARD_WAIT_READY

  RETURN
```

- 原点復帰

```
'=====================================
' ARD 原点復帰
'=====================================
*ARD_HOME

  prc$="ARD_HOME" : PR prc$

  GOSUB *ARD_WRITE_REG_L &H02C2 500              /* 原点復帰速度
  GOSUB *ARD_WRITE_REG_L &H02C6 100              /* 原点復帰起動速度
  GOSUB *ARD_WRITE_REG_L &H02C4 200              /* 原点復帰加減速
  GOSUB *ARD_WRITE_REG_L &H02CA 0                /* 原点復帰開始方向 0:-側
  GOSUB *ARD_WRITE_REG_W &H007D &H10             /* ドライバ入力指令 Home
  GOSUB *ARD_WRITE_REG_W &H007D &H0
  GOSUB *ARD_WAIT_READY
  GOSUB *ARD_CURRENT_POS
  RETURN
```

- ユーザーパラメータの設定

```
'=====================================
' ARD パラメータセット
'=====================================
*ARD_SET_PARAM

  prc$="ARD_SET_PARAM" : PR prc$

  GOSUB *ARD_WRITE_REG_L &H21A 0                 /* HOMES 接点設定 0:N.O.
  GOSUB *ARD_WAIT_READY
  GOSUB *ARD_WRITE_REG_L &H18C 1                 /* Configuration
  GOSUB *ARD_WRITE_REG_L &H18C 0
  GOSUB *ARD_WAIT_READY
  RETURN
```

アイエイアイ ERC2-SA6C-I-PM-6-100-SE-P

　このサンプルは、機能別にサブルーチン化しています。クエリ、レスポンスのデータ数が多いので配列変数を用いています。

変数初期化、タイムアウト設定

```
'=====================================
' 使用する配列変数などを初期化
'=====================================
*QUERY_PREPARE
  FILL ERC2_QUERY(0) 40 0
  FILL ERC2_RES(0) 40 0
  RegSize$=""
  TIME 5                                  /* Silent Interval
  TMOUT 2000                              /* 通信タイムアウト
  RETURN
```

タッチパネル エラー表示

```
'=====================================
' エラー表示
'=====================================
*ERROR_DISP
  oldpage=MBK(2)
  MBK(2)=7                                /* タッチパネルエラー表示画面
  FORMAT ""
  SELECT_CASE r_res                       /* r_resはR_RTUの戻り値
    CASE -1 : errmsg$=prc$+" TIME OUT"
    CASE 0  : errmsg$=prc$+" CRC ERROR"
    CASE_ELSE : errmsg$=prc$+" ?"
  END_SELECT
  S_MBK errmsg$ 360 30
  WAIT SW(71500)==1                       /* Restartボタン
  MBK(2)=oldpage
  TMOUT 2000                              /* 通信タイムアウト再設定
  RETURN
```

タッチパネル 重故障表示

```
'=====================================
' 重故障、軽故障等のエラー表示
'=====================================
*ERC_ERROR_ALMH
_VAR regstat

  MBK(2)=7
  FORMAT "0000"
  errmsg$=prc$+" DSS1=&H"+HEX$(regstat)
  S_MBK errmsg$ 360 30
  END
```

デバイスステータスレジスタ読込

```
'=====================================
' ERC DSS1レジスターチェック
'=====================================
*ERC_CHECK_DSS1
  prc$="ERC_CHECK_DSS1" : PR prc$

  GOSUB *QUERY_PREPARE
  slaveadr=1 : funccode=3
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H9005  /* 開始アドレス
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&H0001  /* レジスタの数

  DO
    X_RTU slaveadr funccode RegSize$ ERC2_QUERY(0)
    r_res=R_RTU(5)
    IF r_res==1 THEN
      R_RTU a c "BW" ERC2_RES(0)
      DSS1_STAT=ERC2_RES(1)
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN
```

原点復帰

```
'=====================================
' ERC 原点復帰
'=====================================
*ERC_HOME
  prc$="ERC_HOME" : PR prc$

  GOSUB *QUERY_PREPARE
  slaveadr=1 : funccode=&H5
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H040B    /* 原点復帰指令
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&HFF00    /* 原点復帰実行
  GOSUB *ERC_HOME_QUERY

  GOSUB *QUERY_PREPARE
  slaveadr=1 : funccode=&H5
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H040B   /* 原点復帰指令
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&H0000   /* 通常状態に戻す
  GOSUB *ERC_HOME_QUERY

  GOSUB *ERC_WAIT_MOVE_END &H10
  RETURN

*ERC_HOME_QUERY
  DO
    X_RTU slaveadr funccode RegSize$ ERC2_QUERY(0)
    r_res=R_RTU(6)
    IF r_res==1 THEN
      R_RTU a c "WW" ERC2_RES(0)
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN
```

絶対位置を指定して移動

```
'====================================
' ERC 直値移動命令(座標移動)
'====================================
*ERC_MOVE_ABS
_VAR dest speed accel
/* Command Format
/* "01109900000912000013880000000A00002710001E00000000"
/*  01 slave address
/*    10 function code
/*      9900 start address
/*          0009 register count
/*              12 byte count
/*                00001388 destination point high 50mm*100=5000 = 0x1388
/*                        0000000A in-position width
/*                                00002710 speed 100mm/sec*100=10000 = 0x2710
/*                                        001E acceleration/deceleration 0.3G*100=30=0x1E
/*                                            0000 press
/*                                                0000 control flag
  prc$="ERC_MOVE_ABS" : PR prc$ dest speed accel

  GOSUB *QUERY_PREPARE

  slaveadr=1 : funccode=&H10
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H9900 /* 開始アドレス
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&H0009 /* レジスタの数
  RegSize$=RegSize$+"B" : ERC2_QUERY(2)=&H12   /* バイト数
  RegSize$=RegSize$+"L" : ERC2_QUERY(3)=dest   /* 目標位置指定
  RegSize$=RegSize$+"L" : ERC2_QUERY(4)=10     /* 位置決め幅指定
  RegSize$=RegSize$+"L" : ERC2_QUERY(5)=speed  /* 速度指定
  RegSize$=RegSize$+"W" : ERC2_QUERY(6)=accel  /* 加減速度指定
  RegSize$=RegSize$+"W" : ERC2_QUERY(7)=0      /* 押し付け時電流制限指定
  RegSize$=RegSize$+"W" : ERC2_QUERY(8)=0      /* 制御フラグ指定 ビット3=0 通常動作

  DO
    X_RTU slaveadr funccode RegSize$ ERC2_QUERY(0)
    r_res=R_RTU(6)
    IF r_res==1 THEN
      R_RTU a c "WW" ERC2_RES(0)
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP

  GOSUB *ERC_WAIT_MOVE_END &H8                    /* &H8=PEND status

  RETURN
```

相対位置を指定して移動

```
'=====================================
' ERC 直値移動命令(相対移動)
'=====================================
*ERC_MOVE_REL
_VAR dest speed accel
/* Command Format
/* "0110990000091200001388000000A00002710001E00000008"
/*  01 slave address
/*    10 function code
/*      9900 start address
/*          0009 register count
/*              12 byte count
/*                00001388 destination point 50mm*100=5000 = 0x1388
/*                        0000000A in-position width
/*                                00002710 speed 100mm/sec*100=10000 = 0x2710
/*                                        001E acceleration/deceleration 0.3G*100=30=0x1E
/*                                            0000 press
/*                                               0008 control flag
  prc$="ERC_MOVE_REL" : PR prc$ dest speed accel

  GOSUB *QUERY_PREPARE

  slaveadr=1 : funccode=&H10
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H9900  /* 開始アドレス:目標位置指定レジスタ
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&H0009  /* レジスタ数
  RegSize$=RegSize$+"B" : ERC2_QUERY(2)=&H0012  /* バイト数
  RegSize$=RegSize$+"L" : ERC2_QUERY(3)=dest    /* 目標位置指定
  RegSize$=RegSize$+"L" : ERC2_QUERY(4)=&H000A  /* 位置決め幅
  RegSize$=RegSize$+"L" : ERC2_QUERY(5)=speed   /* 速度指定
  RegSize$=RegSize$+"W" : ERC2_QUERY(6)=accel   /* 加速度指定
  RegSize$=RegSize$+"W" : ERC2_QUERY(7)=0       /* 押し付け時電流制限指定
  RegSize$=RegSize$+"W" : ERC2_QUERY(8)=&H0008  /* 制御フラグ ビット3=1 インクリメンタル

  DO
    X_RTU slaveadr funccode RegSize$ ERC2_QUERY(0)
    r_res=R_RTU(6)
    IF r_res==1 THEN
      R_RTU a c "WW" ERC2_RES(0)
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP

  GOSUB *ERC_WAIT_MOVE_END &H08                 /* &H8=PEND status

  RETURN
```

ポジション番号を指定して移動

```
'=====================================
' ERC 位置決め動作(点移動)
'=====================================
*ERC_MOVE_POINT
_VAR pointnum

  prc$="ERC_MOVE_POINT" : PR prc$ pointnum

  GOSUB *QUERY_PREPARE
  slaveadr=1 : funccode=&H6
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H0D03    /* ポジション番号指定レジスタ
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=pointnum  /* ポジション番号
  GOSUB *ERC_MOVE_POINT_QUERY                     /* クエリ実行

  GOSUB *QUERY_PREPARE
  slaveadr=1 : funccode=&H5
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H040C    /* 位置決め動作起動命令
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&HFF00    /* 指定位置に移動
  GOSUB *ERC_MOVE_POINT_QUERY

  GOSUB *QUERY_PREPARE
  slaveadr=1 : funccode=&H5
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H040C    /* 位置決め動作起動命令
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&H0000    /* 通常状態に戻す
  GOSUB *ERC_MOVE_POINT_QUERY

  GOSUB *ERC_WAIT_MOVE_END &H8
  RETURN

*ERC_MOVE_POINT_QUERY
  DO
    X_RTU slaveadr funccode RegSize$ ERC2_QUERY(0)
    r_res=R_RTU(6)
    IF r_res==1 THEN
      R_RTU a c "WW" ERC2_RES(0)
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN
```

移動完了待ち

```
'=====================================
' ERC 移動完了待ち (DSS1 Read and Check)
'=====================================
*ERC_WAIT_MOVE_END
_VAR checkbit

  prc$="ERC_WAIT_MOVE_END" : PR prc$ checkbit

  DO
    GOSUB *QUERY_PREPARE

    slaveadr=1 : funccode=3
    RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H9005    /* デバイスステータスレジスタ1
    RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&H0001    /* レジスタの数

    DO
      X_RTU slaveadr funccode RegSize$ ERC2_QUERY(0)
      r_res=R_RTU(5)
      IF r_res==1 THEN
        R_RTU a c "BW" ERC2_RES(0)
        IF ERC2_RES(1)&checkbit==checkbit THEN
          RETURN
        END_IF
        IF ERC2_RES(1)&&H0F00<>0 THEN                /* 重故障、軽故障等のステータス
          GOSUB *ERC_ERROR_ALMH ERC2_RES(1)
        END_IF
        BREAK
      ELSE
        GOSUB *ERROR_DISP
      END_IF
    LOOP

  LOOP

  RETURN
```

ポジションデータ読み込み

```
'=====================================
' ERC 点のパラメータ読込
'=====================================
*ERC_READ_POINT_PARAM
_VAR pointnum

  prc$="ERC_READ_POINT_PARAM" : PR prc$ pointnum

  GOSUB *QUERY_PREPARE

  slaveadr=1 : funccode=3
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H1000+(16*pointnum) /* 開始アドレス
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&H000F               /* レジスタの数

  DO
    X_RTU slaveadr funccode RegSize$ ERC2_QUERY(0)
    r_res=R_RTU(33)
    IF r_res==1 THEN
      R_RTU a c "BLLLLLWWWWW" ERC2_RES(0)
      dtadr=pointnum*40+5000          /* P1:5040~ P2:5080~ P3:5120~ P4:5160~ P5:5200~
      FILL MBK(dtadr) 40 0            /* data clear
      MBK(dtadr~Lng)=pointnum         /* 点番号
      MBK(dtadr+2~Lng)=ERC2_RES(1)    /* PCMD  目標位置
      MBK(dtadr+4~Lng)=ERC2_RES(2)    /* INP   位置決め幅
      MBK(dtadr+6~Lng)=ERC2_RES(3)    /* VCMD  速度指令
      MBK(dtadr+8~Lng)=ERC2_RES(4)    /* ZNMP  個別ゾーン境界+側
      MBK(dtadr+10~Lng)=ERC2_RES(5)   /* ZNLP  個別ゾーン境界-側
      MBK(dtadr+12~Lng)=ERC2_RES(6)   /* ACMD  加速指令
      MBK(dtadr+14~Lng)=ERC2_RES(7)   /* DCMD  減速指令
      MBK(dtadr+16~Lng)=ERC2_RES(8)   /* PPOW  押し付け時電流制限値
      MBK(dtadr+18~Lng)=ERC2_RES(9)   /* LPOW  負荷電流閾値
      MBK(dtadr+20~Lng)=ERC2_RES(10)  /* CTLF  制御フラグ指定
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP

  RETURN
```

ポジションデータの保存

```
'=====================================
' ERC 点のパラメータ書き込み
'=====================================
*ERC_WRITE_POINT_PARAM
_VAR pointnum
/* Command Format
/* 011010C0 000F1E0000271000000000A00004E2000001770000000FA00001001E000000000000
/* 01 slave address
/*   10 function code
/*     10C0 start adress 10C0 = &H1000 + &HC0 , &HC0=192 = 16*12
/*          000F register count
/*              1E byte count
/*                00002710 PCMD
/*                        0000000A INP
/*                                00004E20 VCMD
/*                                        00001770 ZNMP
/*                                                00000FA0 ZNLP
/*                                                        0001 ACMD
/*                                                            001E DCMD
/*                                                                0000 PPOW
/*                                                                    0000 LPOW
/*                                                                        0000 CTLF
  prc$="ERC_WRITE_POINT_PARAM" : PR prc$ pointnum

  GOSUB *QUERY_PREPARE

  slaveadr=1 : funccode=&H10
  dtadr=pointnum*40+5000
  datastart=MBK(dtadr~Lng)*16+&H1000

  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=datastart           /* 開始アドレス
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&H000F              /* レジスタの数
  RegSize$=RegSize$+"B" : ERC2_QUERY(2)=&H001E              /* バイト数
  RegSize$=RegSize$+"L" : ERC2_QUERY(3)=MBK(dtadr+2~Lng)    /* PCMD 目標位置
  RegSize$=RegSize$+"L" : ERC2_QUERY(4)=MBK(dtadr+4~Lng)    /* INP  位置決め幅
  RegSize$=RegSize$+"L" : ERC2_QUERY(5)=MBK(dtadr+6~Lng)    /* VCMD 速度指令
  RegSize$=RegSize$+"L" : ERC2_QUERY(6)=MBK(dtadr+8~Lng)    /* ZNMP 個別ゾーン境界+側
  RegSize$=RegSize$+"L" : ERC2_QUERY(7)=MBK(dtadr+10~Lng)   /* ZNLP 個別ゾーン境界-側
  RegSize$=RegSize$+"W" : ERC2_QUERY(8)=MBK(dtadr+12~Lng)   /* ACMD 加速度指令
  RegSize$=RegSize$+"W" : ERC2_QUERY(9)=MBK(dtadr+14~Lng)   /* DCMD 減速度指令
  RegSize$=RegSize$+"W" : ERC2_QUERY(10)=MBK(dtadr+16~Lng)  /* PPOW 押付け時電流制限値
  RegSize$=RegSize$+"W" : ERC2_QUERY(11)=MBK(dtadr+18~Lng)  /* LPOW 負荷電流閾値
  RegSize$=RegSize$+"W" : ERC2_QUERY(12)=MBK(dtadr+20~Lng)  /* CTLF 制御フラグ指定

  DO
    X_RTU slaveadr funccode RegSize$ ERC2_QUERY(0)
    r_res=R_RTU(6)
    IF r_res==1 THEN
      R_RTU a c "WW" ERC2_RES(0)
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP

  RETURN
```

現在位置をポジションとして保存

```
'=====================================
' ERC ティーチング
'=====================================
*ERC_TEACHING
_VAR pointnum

  prc$="ERC_TEACHING" : PR prc$

  GOSUB *QUERY_PREPARE
  slaveadr=1 : funccode=&H5
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H0414    /* ティーチモード指令
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&HFF00    /* 教示モード
  GOSUB *ERC_TEACHING_QUERY

  GOSUB *QUERY_PREPARE
  slaveadr=1 : funccode=&H6
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H0D03    /* ポジション番号指定レジスタ
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=pointnum  /* ポジション番号
  GOSUB *ERC_TEACHING_QUERY

  GOSUB *QUERY_PREPARE
  slaveadr=1 : funccode=&H5
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H0415    /* ポジションデータ取込指令
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&HFF00    /* 取込
  GOSUB *ERC_TEACHING_QUERY

  TIME 50

  GOSUB *QUERY_PREPARE
  slaveadr=1: funccode=&H5
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H0415    /* ポジションデータ取込指令
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&H0000    /* 通常状態
  GOSUB *ERC_TEACHING_QUERY

  GOSUB *QUERY_PREPARE
  slaveadr=1 : funccode=&H5
  RegSize$=RegSize$+"W" : ERC2_QUERY(0)=&H0414    /* ティーチモード指令
  RegSize$=RegSize$+"W" : ERC2_QUERY(1)=&H0000    /* 通常運転モード
  GOSUB *ERC_TEACHING_QUERY

  RETURN

*ERC_TEACHING_QUERY
  DO
    X_RTU slaveadr funccode RegSize$ ERC2_QUERY(0)
    r_res=R_RTU(6)
    IF r_res==1 THEN
      R_RTU a c "WW" ERC2_RES(0)
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN
```

ADVANTECH ADAM-4055

出力ビット操作、パラレル入力

```
'=====================================
' ADAM ビットON
'=====================================
*ADAM_ON
_VAR bitnum
  prc$="ADAM_ON" : PR prc$ bitnum
  DO
    TIME 5 : TMOUT 2000
    X_RTU 10 5 (bitnum+16)~Wrd &HFF00~Wrd
    r_res=R_RTU(6)
    IF r_res==1 THEN
      R_RTU a c adam_res0~Wrd adam_res1~Wrd
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN

'=====================================
' ADAM ビットOFF
'=====================================
*ADAM_OFF
_VAR bitnum
  prc$="ADAM_OFF" : PR prc$ bitnum
  DO
    TIME 5 : TMOUT 2000
    X_RTU 10 5 (bitnum+16)~Wrd &H0000~Wrd
    r_res=R_RTU(6)
    IF r_res==1 THEN
      R_RTU a c adam_res0~Wrd adam_res1~Wrd
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN

'=====================================
' ADAM パラレル入力
'=====================================
*ADAM_IN
  prc$="ADAM_IN" : PR prc$
  DO
    TIME 5 : TMOUT 2000
    X_RTU 10 1 &H0000~Wrd &H0008~Wrd
    r_res=R_RTU(4)
    IF r_res==1 THEN
      R_RTU a c adam_res0~Wrd adam_res1~Wrd
      BREAK
    ELSE
      GOSUB *ERROR_DISP
    END_IF
  LOOP
  RETURN adam_res0&&HFF
```

送受信バッファの確認

RS コマンドで送受信バッファを Dump します。

オリエンタルモーター ARD-KD に対して

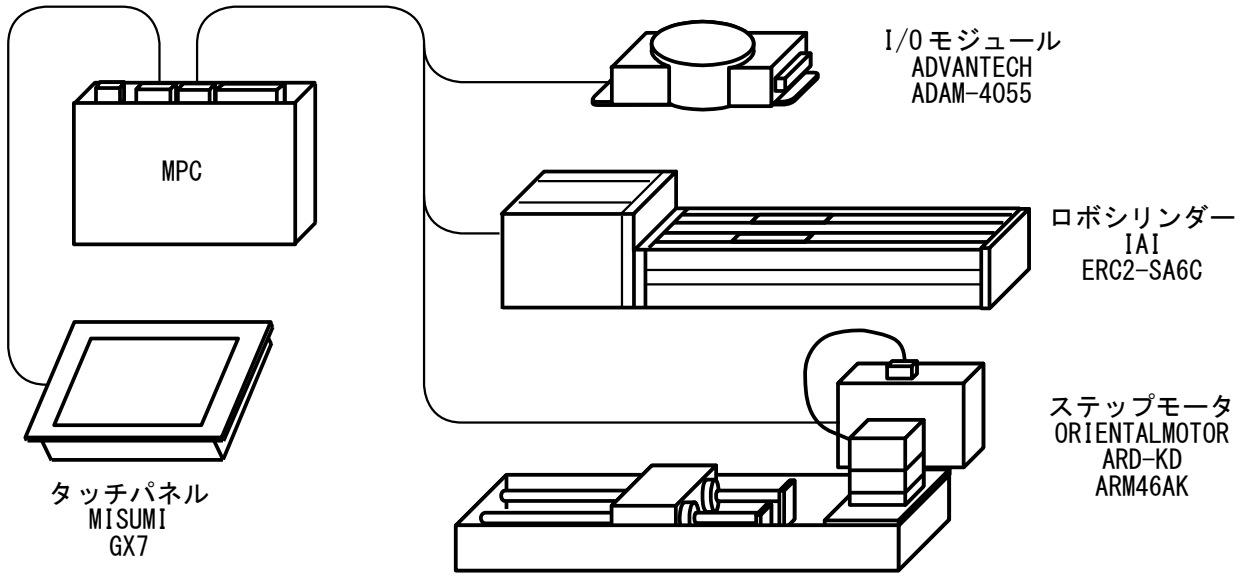
```
X_RTU 2 &H10 &H21A~Wrd 2~Wrd 4~Byt 0    /* 送信 'HOMES の接点設定を N.O.にする' というクエリー
/* SlaveAdd=2,FunctionCode=&H10,StartRegAdd=&H21A,WriteRegCount=2,WriteByteCount=4,WriteDate=0
r_res=R_RTU(6)                          /* 受信
```

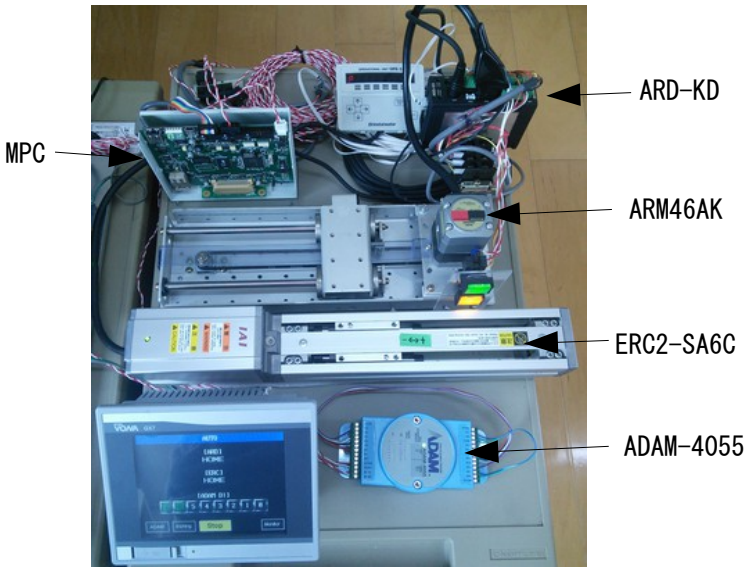
を実行したときの表示例。

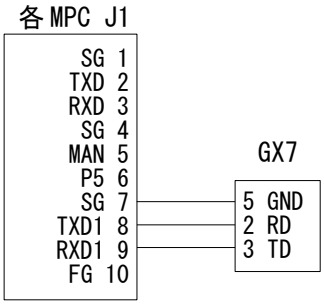
```
#RS 2        /* RS コマンドをダイレクト実行

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><x10> <STX><x1A><NUL><STX>  a  <x84> <---Most recent
                                        0x02 0x10  0x02 0x1A 0x00 0x02 0x61 0x84
                                        スレーブアドレス
                                             ファンクションコード
                                                  レジスタアドレス
                                                       レジスタの数
                                                            CRC

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><STX><x10><STX><x1A><NUL><STX><EOT> <NUL><NUL><NUL><NUL>  d  <xF8> <---Most recent
                  0x02 0x10 0x02 0x1A 0x00 0x02 0x04  0x00 0x00 0x00 0x00 0x64 0xF8
                  スレーブアドレス
                       ファンクションコード
                            書き込みの起点となるレジスタアドレス
                                 起点のレジスタアドレスから書き込むレジスタの数
                                      バイト数
                                           書き込み値
                                                    CRC
```

動作確認

　前述の Modbus RTU 機器をマルチドロップ接続して動作を確認しました。
原点復帰・ポジション移動・パラメータ R/W・ティーチング・出力/入力などを順番に行います。

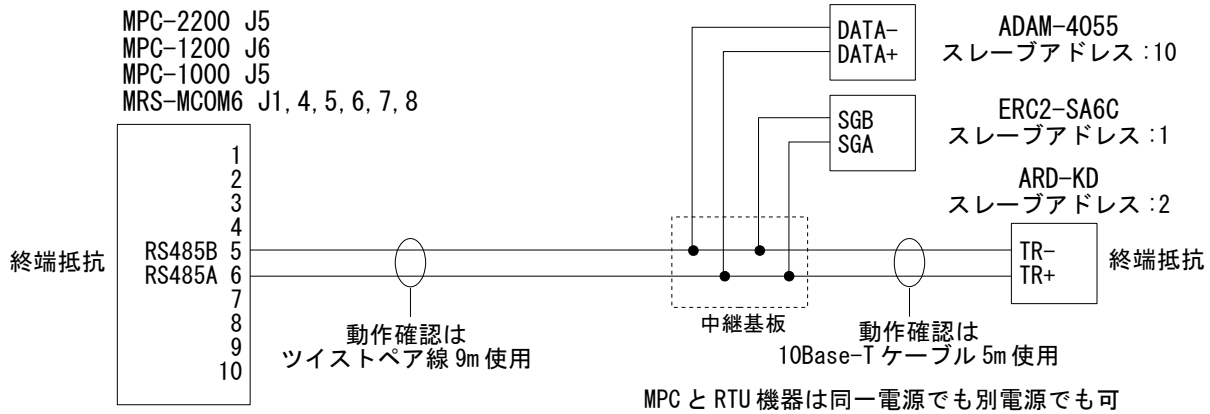






RTU 機器のアドレス、通信は専用ツールや DSW で設定

通信仕様は同一
38400bps, 8bit, PrityNone, 1Stop, XON/XOFFNone

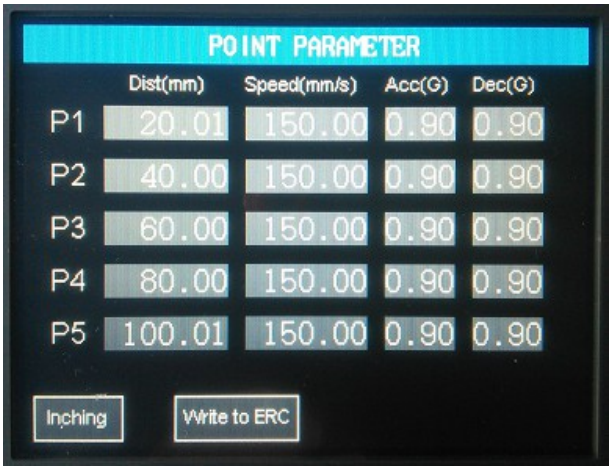


MPC と RTU 機器は同一電源でも別電源でも可

動作確認プログラムのタッチパネル表示

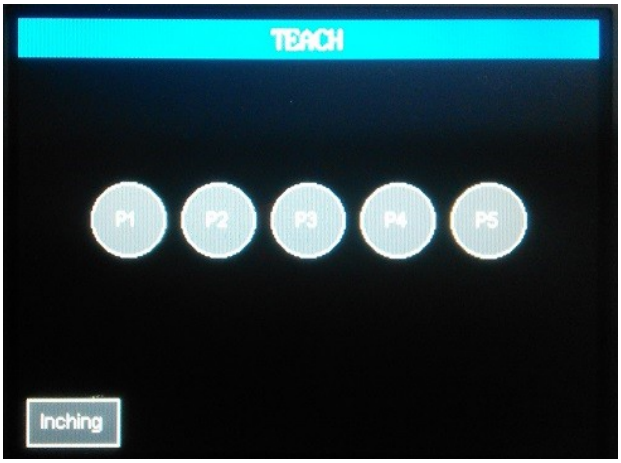
自動運転
ARD 、 ERC2 の各種移動、 ADAM の ON/OFF を繰り返し行います。

インチング
ERC2 を設定した量だけ移動します。



点のパラメータ
ERC2 の点のパラメータを表示します。
変更、書込みもできます。

点移動
ERC2 が指定した点に移動します。



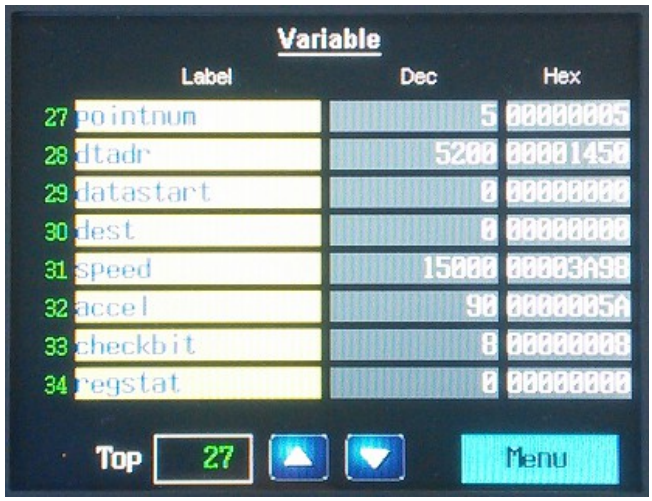
ティーチング
ERC2 の現位置を指定した点として教示します。

ADAM 入出力
ADAM の出力を操作、入力を表示します。

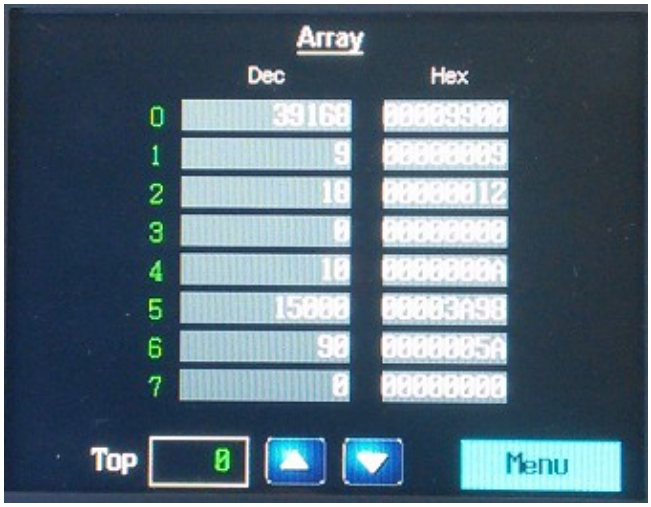
タッチパネルのモニター機能

Modbus 通信とは関係ありませんが、タッチパネルにモニター画面を入れておくと、各種データをリアルタイムで参照・変更できます。サンプルプログラムは自動運転画面の Monitor ボタンで遷移します。

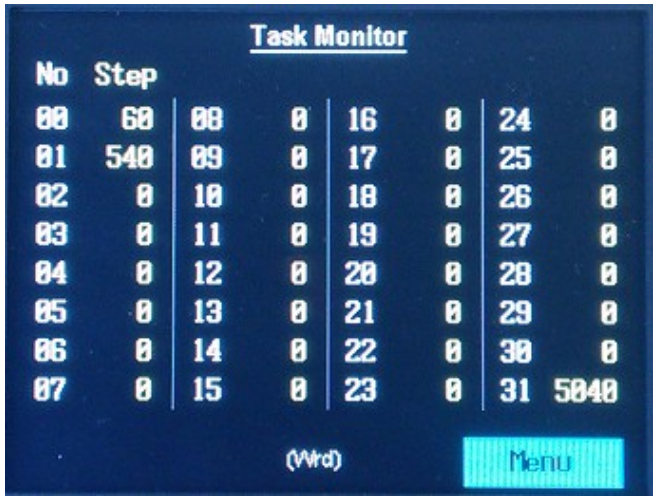
DT(MBK) エリア
サンプルプログラムでは ERC2 の各点のパラメータを DT エリアに格納しています。
点 1:DT5040 ～, 点 2:DT5080 ～, 点 3:DT5120 ～
点 4:DT5160 ～, 点 5:DT5200 ～



変数
プログラムで使用しているグローバル変数の値を参照・変更できます。



配列変数
サンプルプログラムでは
DIM ERC2_QUERY(40)
DIM ERC2_RES(40)
と宣言しているので
ERC2_QUERY が No.0 ～ 39, ERC2_RES が No.40 ～ 79
に対応します。変更もできます。



タスクモニタ
実行中のステップ番号です。

ダウンロード

本文に掲載の MPC プログラムは下記からダウンロードできます。
http://www.departonline.jp/acceleng/dev_uty.php の No.390

参考資料

- 株式会社 アイエイアイ
PCON, ACON,SCIN,ERC2 シリアル通信 【Modbus 版】 取り扱い説明書 第 3 版,第 5 版
- オリエンタルモーター株式会社
αSTEP AR シリーズ DC 電源入力 位置決め機能内蔵タイプ ユーザーズマニュアル