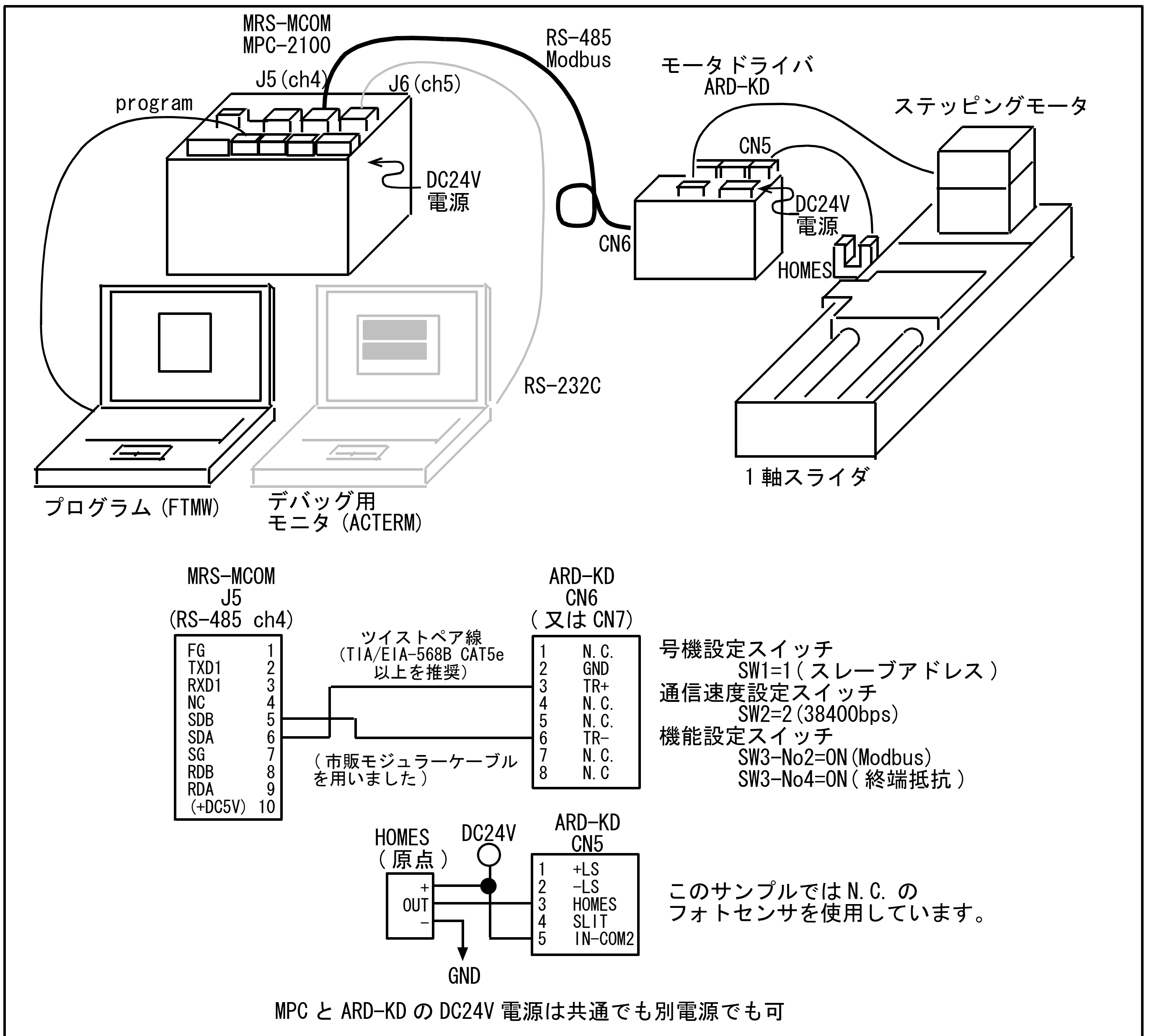
MPC-2100 RS-485 Modbusモード ステップモータ駆動例

2010/07/30

- RS-485 通信機能付きステップモータドライバの使用実験です。
- オリエンタルモーター㈱の ARD-KD(モータとセット)を Modbus RTU モードで制御します。
- 離れた装置にも少ない配線で接続できます(通信だけならケーブル 1 本)。
- この例は MPC-2100 + MRS-MCOM ですが、MPC-2000 + MRS-MCOM や MPC-1000(単体)でも同様な制御が可能です。
- MPC 側(MRS-MCOM の空 RS-232C)にターミナルを接続してデバッグの可視化をしてみました(実行に必要ありません)。
- 参考資料

「オリエンタルモーター㈱ ステッピングモーター αSTEP 高効率 AR シリーズ
DC 電源入力 コントローラ内蔵タイプ (RS-485 通信機能付) ユーザーズマニュアル」

■機器構成



■エラーチェック
Modbus RTU モードのエラーチェックは CRC-16 方式を採用しています。スレーブは受信したメッセージの CRC-16 を計算して、メッセージに含まれるエラーチェックの値と比較します。CRC-16 の計算値とエラーチェックが一致していれば、正常なメッセージと判断します。

◆CRC-16 の計算方法
1. 初期値を FFFFh とし、FFFFh とスレーブアドレス（8 ビット）の排他的論理和（XOR）を計算します。
2. 手順 1 の結果を 1 bit 右へシフトします。このシフトはあふれたビットが「1」になるまで行います。
3. あふれたビットが「1」になったら、手順 2 の結果と A001h の XOR を計算します。
4. シフトが 8 回になるまで、手順 2 と手順 3 を繰り返します。
5. 手順 4 の結果とファンクションコード（8 ビット）の XOR を計算します。
すべてのバイトに対して、手順 2 から 4 を繰り返します。
最後の結果が CRC-16 の計算結果になります。

(出典：オリエンタルモーター㈱ ユーザーズマニュアル)

■MPC プログラム
◆制御手順
基本的には、ARD-KD の必要なレジスタを設定する→ドライバ入力指令(&H007C)のビットを立てる→ドライバ出力指令(&H007E)の READY が立つのを待つ という手順です。

◆サブルーチン概要
*QUERY_WRITE_REG reg data
reg:レジスタアドレス、data:書込みデータ。ARD-KD のレジスタに書込みます(16 ビット)。

*QUERY_WRITE_REG_L reg data
reg:レジスタアドレス、data:書込みデータ。ARD-KD のレジスタに書込みます(32 ビット)。

*QUERY_READ_REG reg
reg:レジスタアドレス。ARD-KD からレジスタを読み込みます(16 ビット)。

*QUERY_READ_REG_L reg
reg:レジスタアドレス。ARD-KD からレジスタを読み込みます(32 ビット)。

*QUERY_SND datacnt
datacnt:データ文字数。ARD-KD にクエリを送信します。

*RES_RCV datacnt
datacnt:データ文字数。ARD-KD からレスポンスを受信します。

*CRC_CALC
送信データ・受信データの CRC-16 計算を行います。

*WAIT_FOR_READY
ARD-KD のレディーが立つまで待ちます。

*ARD_SET_PARAM
ARD-KD の運転前に必要な設定(configuration で反映されるもの)を行います。

*ARD_NV_INITIALIZE
ARD-KD の NV メモリを初期化します(毎回必要なものではありません)

◆動作概要
- *MOVE と *JOG をタスク切り換えで実行します。
- どちらも最初に原点復帰を行い、スライダの往復動作を繰り返します。
- *MOVE は 「位置 No0」～「位置 No7」の 「位置」や「運転速度」等の運転データを設定して PTP 移動を行います。MPC の MOVS コマンドに点を指定して移動するようなイメージです。
- *JOG は「JOG 移動量」を設定して一定量でピッチ送りをします。サンプルでは往復のピッチ・スピードを変えています。MPC の RMVS コマンドで相対移動するようなイメージです。

◆コード

```
*_BEGIN
  DIM QUERY(16)                                 /* using QUERY(1)~
  DIM RES(16)                                   /* using RES(1)~
  DIM BUFF(16)                                  /* using BUFF(1)~

  CNFG# RS485 4 "38400b8pes1NONE"               /* RS485 initialize
  CNFG# 5 "38400b8pes1NONE"                     /* monitor for debug
  TIME 1000

  slave_address=1                               /* slave address

  IF SW(194)==1 THEN                            /* momentary switch
    GOTO *ARD_NV_INITIALIZE                     /* initialize the NV memory
  END_IF

  GOSUB *ARD_SET_PARAM                          /* setting the parameters
  GOSUB *QUERY_WRITE_REG &H7D &H0               /* reset command register

  FORK 1 *TASK_SWITCH
  END

*TASK_SWITCH
  DO
    QUIT 2
    INPUT# 4 CLR_BUF
    IF SW(195)==0 THEN                          /* select switch
      FORK 2 *JOG
      WAIT SW(195)==1
    ELSE
      FORK 2 *MOVE
      WAIT SW(195)==0
    END_IF
  LOOP

  END

*MOVE
  PRINT "MOVE"

  'pos No0~7 set
  FOR I=0 TO 14 STEP 2
    GOSUB *QUERY_WRITE_REG_L &H400+I I*200      /* position
    GOSUB *QUERY_WRITE_REG_L &H480+I 2000       /* speed
    GOSUB *QUERY_WRITE_REG_L &H500+I 1          /* 0=incremental, 1=absolute
    GOSUB *QUERY_WRITE_REG_L &H580+I 0          /* 0=single, 1=join
    GOSUB *QUERY_WRITE_REG_L &H600+I 2000       /* acceleration
    GOSUB *QUERY_WRITE_REG_L &H680+I 2000       /* deceleration
  NEXT I

  GOSUB *HOME

  DO
    FOR I=0 TO 7
      GOSUB *QUERY_WRITE_REG &H7D &H8|I         /* start or M2,1,0
      GOSUB *QUERY_WRITE_REG &H7D &H0
      GOSUB *WAIT_FOR_READY
      GOSUB *READ_CURRENT_POS
      TIME 100
    NEXT I

    FOR I=7 TO 0 STEP -1
      GOSUB *QUERY_WRITE_REG &H7D &H8|I         /* start or M2,1,0
      GOSUB *QUERY_WRITE_REG &H7D &H0
      GOSUB *WAIT_FOR_READY
      GOSUB *READ_CURRENT_POS
      TIME 100
    NEXT I

  LOOP

END

*JOG
```

```
  PRINT "JOG"
  GOSUB *HOME

  GOSUB *QUERY_WRITE_REG_L &H0288 2000          /* JOG acceleration/deceleration
  GOSUB *QUERY_WRITE_REG_L &H028A 100           /* JOG start speed
  DO

    GOSUB *QUERY_WRITE_REG_L &H0286 500         /* JOG speed Hz
    GOSUB *QUERY_WRITE_REG_L &H1048 400         /* JOG pulse count
    FOR LP_CNT=1 TO 5
      GOSUB *QUERY_WRITE_REG &H7D &H1000        /* +JOG
      GOSUB *QUERY_WRITE_REG &H7D &H0
      GOSUB *WAIT_FOR_READY
      TIME 500
    NEXT LP_CNT

    GOSUB *QUERY_WRITE_REG_L &H0286 1000        /* JOG speed Hz
    GOSUB *QUERY_WRITE_REG_L &H1048 200         /* JOG pulse count
    FOR LP_CNT=1 TO 10
      GOSUB *QUERY_WRITE_REG &H7D &H2000        /* -JOG
      GOSUB *QUERY_WRITE_REG &H7D &H0
      GOSUB *WAIT_FOR_READY
      TIME 500
    NEXT LP_CNT

  LOOP

  END

*HOME
  GOSUB *QUERY_WRITE_REG_L &H2C6 1000           /* HOME Speed
  GOSUB *QUERY_WRITE_REG_L &H2C4 2000           /* HOME acceleration/deceleration
  'GOSUB *QUERY_WRITE_REG_L &H21A 1             /* HOMES N.C. -> set in *ARD_SET_PARAM
  GOSUB *QUERY_WRITE_REG_L &H2CA 1              /* home dir

  GOSUB *QUERY_WRITE_REG &H7D &H10              /* HOME
  GOSUB *QUERY_WRITE_REG &H7D &H0
  GOSUB *WAIT_FOR_READY
  GOSUB *READ_CURRENT_POS
  'GOSUB *P-PRESET                              /* will be not necessary
RETURN

*P-PRESET
  GOSUB *QUERY_WRITE_REG_L &H18A 1              /* P-PRESET
  GOSUB *QUERY_WRITE_REG_L &H18A 0
  GOSUB *WAIT_FOR_READY
  GOSUB *READ_CURRENT_POS
RETURN

*READ_CURRENT_POS                               /* current position read
  GOSUB *QUERY_READ_REG_L &HCC
  current_pos=(RES(4)<<24)+(RES(5)<<16)+(RES(6)<<8)+RES(7)
  PRINT "current_pos=" current_pos
RETURN

*WAIT_FOR_READY
  DO
    GOSUB *QUERY_READ_REG &H7F
    IF RES(5)&&H20<>0 THEN                      /* ready bit
      BREAK
    END_IF
  LOOP
RETURN

*ARD_SET_PARAM
  GOSUB *QUERY_WRITE_REG_L &H21A 1              /* HOMES N.C.
  GOSUB *WAIT_FOR_READY

  GOSUB *QUERY_WRITE_REG_L &H18C 1              /* configuration
  GOSUB *QUERY_WRITE_REG_L &H18C 0
  GOSUB *WAIT_FOR_READY

RETURN

*ARD_NV_INITIALIZE
  PRINT "initialize the NV memory"
```

```
  GOSUB *QUERY_WRITE_REG_L &H18E 1
  GOSUB *QUERY_WRITE_REG_L &H18E 0
  GOSUB *WAIT_FOR_READY

END

*ARD_ALARM_RESET
  GOSUB *QUERY_WRITE_REG_L &H180 1
  GOSUB *QUERY_WRITE_REG_L &H180 0

RETURN

'===============
' 4byte write
'===============
*QUERY_WRITE_REG_L
  _VAR REG_ADD WRITE_DATA
  'PRINT "== WRITE ==" HEX$(REG_ADD) HEX$(WRITE_DATA)

  QUERY(1)=slave_address                        /* slave address
  QUERY(2)=&H10                                 /* write to register
  QUERY(3)=REG_ADD/256                          /* begining to write H
  QUERY(4)=REG_ADD&&HFF                         /* begining to write L
  QUERY(5)=0                                    /* write count H
  QUERY(6)=2                                    /* write count L
  QUERY(7)=4                                    /* write count * 2
  QUERY(8)=(WRITE_DATA>>24)&&HFF                /* write data HH
  QUERY(9)=(WRITE_DATA>>16)&&HFF                /* write data HL
  QUERY(10)=(WRITE_DATA>>8)&&HFF                /* write data LH
  QUERY(11)=WRITE_DATA&&HFF                     /* write data LL

  DO                                            /* if CRC error then retry
    GOSUB *QUERY_SND 11
    GOSUB *RES_RCV 6
    IF CRC_OK==1 THEN : BREAK : END_IF
  LOOP

RETURN

'===============
' 2byte write
'===============
*QUERY_WRITE_REG
  _VAR REG_ADD WRITE_DATA
  'PRINT "== WRITE ==" HEX$(REG_ADD) HEX$(WRITE_DATA)

  QUERY(1)=slave_address                        /* slave address
  QUERY(2)=&H06                                 /* write to register
  QUERY(3)=REG_ADD/256                          /* begining to write H
  QUERY(4)=REG_ADD&&HFF                         /* begining to write L
  QUERY(5)=(WRITE_DATA>>8)&&HFF                 /* write data H
  QUERY(6)=WRITE_DATA&&HFF                      /* write data L

  DO                                            /* if CRC error then retry
    GOSUB *QUERY_SND 6
    GOSUB *RES_RCV 6
    IF CRC_OK==1 THEN : BREAK : END_IF
  LOOP

RETURN

'===============
' 4byte read
'===============
*QUERY_READ_REG_L
  _VAR REG_ADD
  'PRINT "== READ ==" HEX$(REG_ADD)

  QUERY(1)=slave_address                        /* slave address
  QUERY(2)=&H03                                 /* read from register
  QUERY(3)=REG_ADD/256                          /* begining to read H
  QUERY(4)=REG_ADD&&HFF                         /* begining to read L
  QUERY(5)=&H00                                 /* read count H
  QUERY(6)=&H02                                 /* read count L

  DO                                            /* if CRC error then retry
```

```
    GOSUB *QUERY_SND 6
    GOSUB *RES_RCV 7
    IF CRC_OK==1 THEN : BREAK : END_IF
  LOOP

RETURN

'===============
' 2byte read
'===============
*QUERY_READ_REG
  _VAR REG_ADD
  'PRINT "== READ ==" HEX$(REG_ADD)

  QUERY(1)=slave_address                        /* slave address
  QUERY(2)=&H03                                 /* read from register
  QUERY(3)=REG_ADD/256                          /* begining to read H
  QUERY(4)=REG_ADD&&HFF                         /* begining to read L
  QUERY(5)=&H00                                 /* read count H
  QUERY(6)=&H01                                 /* read count L

  CRC_OK=0

  DO                                            /* if CRC error then retry
    GOSUB *QUERY_SND 6
    GOSUB *RES_RCV 5
    IF CRC_OK==1 THEN : BREAK : END_IF
  LOOP

RETURN

'===============
' send query
'===============
*QUERY_SND
  _VAR DATA_CNT
  'PRINT "SEND"

  DIMCPY QUERY(1) BUFF(1) DATA_CNT              /* QUERY()->BUFF()
  GOSUB *CRC_CALC
  QUERY(DATA_CNT+1)=CRC_L
  QUERY(DATA_CNT+2)=CRC_H

  SEND$=""                                      /* The each string size is 256bytes
  str_pt=SEND$                                  /* str_pt is the pointer of SEND$
  FOR array_num=1 TO DATA_CNT+2
    POKE QUERY(array_num) str_pt                /* binary data create
    INC str_pt
    'PRINT "QUERY(" array_num ")=" HEX$(QUERY(array_num))
  NEXT array_num

  PRINT# 4 STR_LEN|DATA_CNT+2 SEND$
  PRINT# 5 STR_LEN|DATA_CNT+2 SEND$             /* monitor for debug

RETURN

'===============
' read response
'===============
*RES_RCV
  _VAR DATA_CNT
  'PRINT "RCV"
  FOR array_num=1 TO DATA_CNT+2
    INPUT# 4 CHR_C|1 RES$
    PRINT# 5 STR_LEN|1 RES$                     /* monitor for debug
    RES(array_num)=ASC(RES$)
    'PRINT "RES(" array_num ")=" HEX$(RES(array_num))
  NEXT array_num

  DIMCPY RES(1) BUFF(1) DATA_CNT                /* RES()->BUFF()
  GOSUB *CRC_CALC
  'PRINT "CRC L H" HEX$(CRC_L) HEX$(CRC_H)

  CRC_OK=1
  IF RES(DATA_CNT+1)<>CRC_L THEN                /* CRC compare RES to CACL
    PRINT "CRC L ERROR"
```

```
    INPUT# 4 CLR_BUF
    CRC_OK=0
    'END
  END_IF
  IF RES(DATA_CNT+2)<>CRC_H THEN
    PRINT "CRC H ERROR"
    INPUT# 4 CLR_BUF
    CRC_OK=0
    'END
  END_IF

RETURN

'===============
' CRC16 calculate
'===============
*CRC_CALC

'== SAMPLE1 ==
  CRC=&HFFFF
  FOR crc_i=1 TO DATA_CNT
    crc_next=BUFF(crc_i)
    CRC=(CRC^crc_next)&&HFFFF
    FOR crc_j=1 TO 8
      crc_carry=CRC&&H1
      CRC=CRC/2
      IF crc_carry=1 THEN
        CRC=(CRC^&HA001)&&HFFFF
      END_IF
      SWAP
    NEXT crc_j
    CRC_L=CRC&&HFF
    CRC_H=(CRC&&HFF00)/256
  NEXT crc_i
RETURN

'== SAMPLE2 ==
/*  CRC=&HFFFF
/*  FOR crc_i=1 TO DATA_CNT
/*    CRC=CRC^BUFF(crc_i)
/*    FOR crc_j=1 TO 8
/*      crc_carry=CRC&&H1
/*      IF CRC<0 THEN
/*        crc_ch=1
/*      ELSE
/*        crc_ch=0
/*        GOTO *CRC_CALC_1
/*      END_IF
/*      CRC=CRC&&H7FFF
/*  *CRC_CALC_1
/*      CRC=CRC/2  /* CRC=INT(CRC/2)
/*      IF crc_ch=1 THEN
/*        CRC=CRC|&H4000
/*      END_IF
/*      IF crc_carry=1 THEN
/*        CRC=CRC^&HA001
/*      END_IF
/*    NEXT crc_j
/*    SWAP
/*  NEXT crc_i
/*  CRC_L=CRC&&HFF
/*  CRC_H=((CRC&&HFF00)/256&&HFF)
/*RETURN
```

■デバッグ用モニタの画面

```
ACTERM
ファイル(F) ウインドウ(W) 編集(E) ヘルプ(H)
~01~03~00~7F~00
~01~B5~D2
~01~03~02~40~20~88~5C
~01~06~00~7D~20~00~00~12
~01~06~00~7D~20~00~00~12
~01~06~00~7D~00~00~19~D2
~01~06~00~7D~00~00~19~D2
~01~03~00~7F~00
~01~B5~D2
~01~03~02~20~00~A1~84
~01~03~00~7F~00
~01~B5~D2
~01~03~02~20~00~A1~84
~01~03~00~7F~00
~01~B5~D2
~01~03~02~20~00~A1~84
~01~03~00~7F~00
~01~B5~D2
~01~03~02~20~00~A1~84
~01~03~00~7F~00
~01~B5~D2
~01~03~02~40~20~88~5C
```

送信クエリ（-JOG コマンド）
"スレーブ=01, 書込=06, アドレス=007D, データ=2000, CRC 下位, CRC 上位"
これは「ドライバ入力指令」の「-JOG」ビットを立てる

レスポンス。正常なら送信クエリと同じ

送信クエリ（コマンドリセット）
"スレーブ=01, 書込=06, アドレス=007D, データ=0000, CRC 下位, CRC 上位"
これは「ドライバ入力指令」の「-JOG」ビットを降ろす

レスポンス。正常なら送信クエリと同じ

送信クエリ（状態確認）
"スレーブ=01, 読出=03, アドレス=007F, レジスタ数=0001, CRC 下位, CRC 上位"
これは「ドライバ出力指令」を読む

レスポンス
"スレーブ=01, 読出=03, データバイト数=02, データ=2000, CRC 下位, CRC 上位"
これは「ドライバ出力指令」の「MOVE」ビットが立っている=移動中

送信クエリ（状態確認）
同上

レスポンス
"スレーブ=01, 読出=03, データバイト数=02, データ=4020, CRC 下位, CRC 上位"
これは「ドライバ出力指令」の「END」と「READY」ビットが立っている
=ドライバが READY 状態

Start(S) Stop(S)
Char Count I 15465 C O 0 C
Monitor Clear(C) Hex(X) Log(L) NL 01 F B
LocalEcho(I) N Y
Emulation(M) ANSI ANSIBBS VT100 VT52
Com(P) 4
Baud(B) 1200 2400 4800 9600 19200 38400 57600 115200
DataBit(D) 7 8
StopBit(G) 1 2
Parity(J) None Even Odd
H/S(K) None Xon/Xoff RTS/CTS Both
EnterKey(Y) CR NL
History(V)
LoopBack(T)

通信内容が見えます。受信バッファは MPC の RS コマンドでも確認できます。