```
/************************************************************************************/
/***                  シリアルインタフェースSCI0（RS-485）                        ***/
/************************************************************************************/

#include    <machine.h>
#include    "typedefine.h"
#include    "iodefine.h"

#include    "support.h"
#include    "sci0_rs485.h"


#pragma section
#pragma bit_order right

/************************************************************************************/

#define SCI0_RTS PORT2.DR.BIT.B2
#define ON  (1)
#define OFF (0)

/*  ステータスフラグの構成  ********************************************************/

typedef union {
    BYTE        BYTE;
    struct{
        BYTE    CHAR2TIME:1;            /* B0:２バイト分以上回線が空いた */
        BYTE    CHAR4TIME:1;            /* B1:４バイト分以上回線が空いた */
        BYTE    RXERR:1;                /* B2:受信異常あり */
        BYTE    TXERR:1;                /* B3:送信異常あり */
        BYTE    RXCOMPL:1;              /* B4:受信完了 */
        BYTE    TXCOMPL:1;              /* B5:送信完了 */
        BYTE    RECEIVED:1;             /* B6:受信あり */
        BYTE    SENDMODE:1;             /* B7:送信モード   */
    } BIT;
} COMMSTAT;
static volatile COMMSTAT sci0_stat;     /* ステータスフラグ */

/************************************************************************************/
/*  送受信用バッファ＆ポインタ                                                      */

#define RX0BUFFSZ   (1024)
static WORD tx0_size,                   /* 送信データサイズ   */
            tx0_rptr,                   /* 送信リードポインタ */
            tx0_cptr;                   /* 送信確認用ポインタ */
static BYTE *tx0_addr;                  /* 送信データアドレス */
static WORD rx0_wptr,                   /* 受信ライトポインタ */
            rx0_size;                   /* 受信フレームバイト数*/
static BYTE rx0_buff[RX0BUFFSZ];        /* 受信バッファ       */


/************************************************************************************/
/* 注意 : PCLK=48MHzであること。                                                    */

/* 通信速度(CKS,BRR)設定用テーブル（SEMR.ABCS=1であること）   */
typedef struct {
    BYTE    CKSn;
    BYTE    BRRN;
} SCIaMODE;
static const SCIaMODE ModeCodeTable[18]={
/* CKSn,BRRN */
    {2,  155},          /*  0:   1200bps( 0.160%) */
    {2,   77},          /*  1:   2400bps( 0.160%) */
    {1,  155},          /*  2:   4800bps( 0.160%) */
    {1,   77},          /*  3:   9600bps(-0.160%) */
    {0,  155},          /*  4:  19200bps( 0.160%) */
    {0,   95},          /*  5:  31250bps( 0.000%) */
    {0,   77},          /*  6:  38400bps( 0.160%) */
    {0,   38},          /*  7:  76800bps( 0.160%) */
    {0,   29},          /*  8: 100000bps */
    {0,   24},          /*  9: 120000bps */
    {0,   19},          /* 10: 150000bps */
    {0,   14},          /* 11: 200000bps */
    {0,   11},          /* 12: 250000bps */
    {0,    9},          /* 13: 300000bps */
    {0,    5},          /* 14: 500000bps */
    {0,    4},          /* 15: 600000bps */
    {0,    3},          /* 16: 750000bps */
    {0,    2},          /* 17:1000000bps */
};

/* 約10byteの時間でコンペアマッチＡを発生させるMTU1.TGRAの値（２バイト分の時間）  */
static const WORD FreeCancelCnt[18]={
    83333L * 2 / 53,        /*  0:   1200bps(8333.3uS/byte) */
```

```
 85     41666L * 2 / 53,         /*  1:   2400bps(4166.6uS/byte) */
 86     20833L * 2 / 53,         /*  2:   4800bps(2083.3uS/byte) */
 87     10416L * 2 / 53,         /*  3:   9600bps(1041.6uS/byte) */
 88      5208L * 2 / 53,         /*  4:  19200bps(520.83uS/byte) */
 89      3200L * 2 / 53,         /*  5:  31250bps(320.00uS/byte) */
 90      2604L * 2 / 53,         /*  6:  38400bps(260.42uS/byte) */
 91      1302L * 2 / 53,         /*  7:  76800bps(130.21uS/byte) */
 92      1000L * 2 / 53,         /*  8: 100000bps(100.0uS/Byte)  */
 93       833L * 2 / 53,         /*  9: 120000bps(83.33uS/Byte)  */
 94       667L * 2 / 53,         /* 10: 150000bps(66.66uS/Byte)  */
 95       500L * 2 / 53,         /* 11: 200000bps(50.00uS/Byte)  */
 96       400L * 2 / 53,         /* 12: 250000bps(40.00uS/Byte)  */
 97       333L * 2 / 53,         /* 13: 300000bps(33.33uS/Byte)  */
 98       200L * 2 / 53,         /* 14: 500000bps(20.00uS/Byte)  */
 99       167L * 2 / 53,         /* 15: 600000bps(16.66uS/Byte)  */
100       133L * 2 / 53,         /* 16: 750000bps(13.33uS/Byte)  */
101          2 * 2               /* 17:1000000bps(10.00uS/Byte)  */
102 };      /*       ↑53=5.333uS単位 */
103
104
105 /*****************************************************************************/
106 /*                          SCI0：タイマー割り込み処理                       */
107 /*****************************************************************************/
108 /* ※10mS周期（MCLK0単位）割り込みで呼ぶこと。                                */
109 static volatile WORD wTmrRxWait;
110 static volatile WORD wTmrTxWait;
111
112 void Timer10mS_sci0(void)
113 {
114     if(wTmrRxWait != 0)
115         --wTmrRxWait;
116     if(wTmrTxWait != 0)
117         --wTmrTxWait;
118 }
119
120 /*****************************************************************************/
121 /*          シリアルコミュニケーションインタフェース(SCI0)初期化             */
122 /*****************************************************************************/
123 /* 引数 : nSpeed = 通信速度(bps)下記参照                                     */
124 /*      :           0:  1200bps    8:100000bps                               */
125 /*      :           1:  2400bps    9:120000bps                               */
126 /*      :           2:  4800bps   10:150000bps                               */
127 /*      :           3:  9600bps   11:200000bps                               */
128 /*      :           4: 19200bps   12:250000bps                               */
129 /*      :           5: 31250bps   13:300000bps                               */
130 /*      :           6: 38400bps   14:500000bps                               */
131 /*      :           7: 76800bps   15:600000bps                               */
132 /* 戻値 : なし                                                               */
133
134 void Init_SCI0(int nSpeed)
135 {
136     /* sci0ステータスクリア */
137     sci0_stat.BYTE = 0;
138     sci0_stat.BIT.TXCOMPL = 1;
139     /* CRCのモジュールストップ解除 */
140     MSTP(CRC) = 0;                      /* SYSTEM.MSTPCRB.BIT.MSTPB23 = 0; */
141
142     /*** シリアルコミュニケーションインタフェース(SCIa)初期化 ***/
143
144     /* SCI0のモジュールストップ解除 */
145     MSTP(SCI0) = 0;                     /* SYSTEM.MSTPCRB.BIT.MSTPB31 = 0; */
146     /* SCI0の送受信動作停止 */
147     SCI0.SCR.BYTE = 0x00;               /* シリアルコントロールレジスタ */
148     /* SCI0のポート設定 */
149     PORT2.ICR.BIT.B1 = 1;               /* P21(RxD0)の入力バッファ有効 */
150     PORT2.DR.BIT.B2  = 0;               /* SCI0_RTS=OFF */
151     PORT2.DDR.BIT.B2 = 1;               /* P22(RTS)を出力ポートに設定  */
152     /* SCI0の初期化 */
153     SCI0.SMR.BYTE = (BYTE)(0x20 | ModeCodeTable[nSpeed].CKSn);  /* 調歩同期式モード (Data8,Even,1Stop)
 */
154     SCI0.SSR.BYTE = 0xC0;               /* ステータスレジスタクリア */
155     SCI0.SCMR.BYTE = 0xF2;              /* シリアルコミュニケーションインタフェースモード、LSBファースト
 */
156     SCI0.SEMR.BYTE = 0x10;              /* 基本クロック8サイクルの期間が1ビット期間の転送レート */
157     SCI0.BRR = ModeCodeTable[nSpeed].BRRN;  /* ビットレート設定 */
158     nop();
159     nop();
160     SCI0.SCR.BYTE = 0xF4;               /* 送信／受信許可、送信／受信／送信完了割り込み許可、 */
161                                         /* 内蔵ボーレートジェネレータ使用                     */
162
163     /*** マルチファンクションタイマパルスユニット(MTU)初期化 ***/
164
165     /* MTU1のモジュールストップ解除 */
166     MSTP(MTU1) = 0;                 /* SYSTEM.MSTPCRA.BIT.MSTPA9 = 0; (MTU0～MTU5共通)  */
```

```
167     /* マルチファンクションタイマパルスユニット(MTU1)初期化 */
168     MTUA.TSTR.BIT.CST1 = 0;         /* MTU1カウント停止 */
169     MTU1.TCR.BYTE = 0x26;           /* TGRAのコンペアマッでTCNTクリア、PCLK/256でカウント */
170     MTU1.TMDR.BYTE = 0x00;          /* 通常動作 */
171     MTU1.TIOR.BYTE = 0x00;          /* MTIOC1A端子出力禁止、MTIOC1B端子出力禁止 */
172     MTU1.TIER.BYTE = 0x03;          /* TGIA割り込み許可、TGIB割り込み許可、TCIV割り込み禁止 */
173     MTUA.TSYR.BIT.SYNC1 = 0;        /* MTU1.TCNTは独立して動作 */
174     MTU1.TCNT = 0x0000;             /* タイマカウンタ */
175     MTU1.TGRA = (WORD)(FreeCancelCnt[nSpeed]*2);    /* タイマジェネラルレジスタA（４バイト分の時間） */
176     MTU1.TGRB = (WORD)(FreeCancelCnt[nSpeed]);      /* タイマジェネラルレジスタB（２バイト分の時間） */
177     /* MTU1カウントスタート */
178     MTUA.TSTR.BIT.CST1 = 1;
179
180
181     /* SCI0割り込み優先順位セット */
182     IPR(SCI0, ) = 3;                /* ERI0,RXI0,TXI0,TEI0共通 */
183     /* SCI0割り込み要求ステータスフラグクリア */
184     IR(SCI0, ERI0) = 0;
185     IR(SCI0, RXI0) = 0;
186 //  IR(SCI0, TXI0) = 0;
187 //  IR(SCI0, TEI0) = 0;
188     /* SCI0受信割り込み許可 */
189     IEN(SCI0, ERI0) = 1;            /* 受信エラー割り込み */
190     IEN(SCI0, RXI0) = 1;            /* 受信割り込み       */
191     /* SCI0送信割り込み禁止 */
192     IEN(SCI0, TXI0) = 0;            /* 送信割り込み       */
193     IEN(SCI0, TEI0) = 0;            /* 送信完了割り込み   */
194
195     /* MTU1割り込み優先順位セット */
196     IPR(MTU1, TGIA1) = 3;           /* コンペアマッチＡ～Ｂ共通 */
197     /* MTU1割り込み要求ステータスフラグクリア */
198     IR(MTU1, TGIA1) = 0;            /* コンペアマッチＡ */
199     IR(MTU1, TGIB1) = 0;            /* コンペアマッチＢ */
200     /* MTU1コンペアマッチ割り込み許可 */
201     IEN(MTU1, TGIA1) = 1;           /* コンペアマッチＡ */
202     IEN(MTU1, TGIB1) = 1;           /* コンペアマッチＢ */
203
204 }
205
206 /*****************************************************************************/
207 /*                        MTU1コンペアマッチ割り込み処理                     */
208 /*****************************************************************************/
209
210 /* コンペアマッチＡ割り込み  ************************************************/
211
212 #pragma interrupt (Int_MTU1_TGIA1)
213 void Int_MTU1_TGIA1(void)
214 {
215     /* 送信ディセーブル中 */
216     if(SCI0.SCR.BIT.TE == 0)
217         return;
218     /* ４バイト分以上回線が空いたことを示す */
219     sci0_stat.BIT.CHAR4TIME = 1;
220 }
221
222 /* コンペアマッチＢ割り込み  ************************************************/
223
224 #pragma interrupt (Int_MTU1_TGIB1)
225 void Int_MTU1_TGIB1(void)
226 {
227     /* 送信ディセーブル中 */
228     if(SCI0.SCR.BIT.TE == 0)
229         return;
230     /* ２バイト分以上回線が空いたことを示す */
231     sci0_stat.BIT.CHAR2TIME = 1;
232     /* これまでの受信をキャンセルする */
233     rx0_wptr = 0;
234 }
235
236 /*****************************************************************************/
237 /*                    受信／受信エラー割り込み処理(SCI0)                     */
238 /*****************************************************************************/
239
240 /* 受信エラー割り込み(SCI0_ERI0)  *******************************************/
241
242 #pragma interrupt (Int_SCI0_ERI0(enable))
243 void Int_SCI0_ERI0(void)
244 {
245     BYTE    byStat;
246
247     /* RDRをダミーリード */
248     SCI0.RDR;
249     /* 回線空き検知タイマをクリア */
250     MTU1.TCNT = 0x0000;             /* タイマカウンタ */
```

```
251     /* フラグクリアで割り込み要求クリア */
252     byStat = SCI0.SSR.BYTE;
253     SCI0.SSR.BYTE = (BYTE)(byStat & 0xC7 | 0xC0);
254     if((SCI0.SSR.BYTE & 0x38) != 0){
255         SCI0.SSR.BYTE = (BYTE)(byStat & 0xC7 | 0xC0);
256     }
257     sci0_stat.BIT.RXERR = 1;
258 }
259
260 /*  受信割り込み(SCI0_RXI0)  ************************************************/
261
262 #pragma interrupt (Int_SCI0_RXI0(enable))
263 void Int_SCI0_RXI0(void)
264 {
265     volatile COMMSTAT   *pStat;
266     BYTE        byRxD;
267
268     /* ステータスフラグのアドレス取得 */
269     pStat = &sci0_stat;
270     /* 回線空き検知タイマをクリア */
271     MTU1.TCNT = 0x0000;
272     pStat->BIT.CHAR2TIME = 0;
273     pStat->BIT.CHAR4TIME = 0;
274
275     /* データ受信 */
276     byRxD = SCI0.RDR;
277     /* 受信モード時 */
278     if(pStat->BIT.SENDMODE == 0){
279         pStat->BIT.RXERR    = 0;                /* 受信正常を示す */
280         pStat->BIT.RECEIVED = 1;                /* 受信ありを示す */
281
282         if((rx0_wptr < RX0BUFFSZ) && (pStat->BIT.RXCOMPL == 0)){
283             /* 受信データ格納 */
284             rx0_buff[rx0_wptr++] = byRxD;
285             /* １バイト目 */
286             if(rx0_wptr == 1){
287                 rx0_size = (WORD)byRxD;         /* バイト数の上位 */
288             }
289             /* ２バイト目 */
290             else if(rx0_wptr == 2){
291                 rx0_size = (WORD)((rx0_size << 8) | (WORD)byRxD);   /* バイト数の下位 */
292                 rx0_size += 4;                  /* データバイト数＋CRCバイト数加算 */
293             }
294             /* ３バイト目以降 */
295             else if(rx0_wptr == rx0_size){
296                 pStat->BIT.RXCOMPL = 1;         /* 受信完了を示す */
297             }
298         }
299     }
300     /* 送信モード時 */
301     else{
302         /* 送信異常 */
303         if(*(tx0_addr + tx0_cptr) != byRxD){
304             IEN(SCI0, TXI0) = 0;                /* 送信割り込み禁止 */
305             SCI0_RTS = OFF;                     /* RTS=OFF */
306             pStat->BIT.SENDMODE = 0;            /* 送信モード終わり */
307             pStat->BIT.TXERR    = 1;            /* 送信異常を示す   */
308             pStat->BIT.TXCOMPL  = 1;            /* 送信完了を示す   */
309         }
310         /* 送信正常 */
311         else{
312             pStat->BIT.TXERR = 0;               /* 送信正常を示す */
313             tx0_cptr++;
314             if(tx0_cptr == tx0_size){
315                 pStat->BIT.SENDMODE = 0;        /* 送信モード終わり */
316                 pStat->BIT.TXCOMPL  = 1;        /* 送信完了を示す   */
317             }
318         }
319     }
320 }
321
322
323 /*  送信割り込み(SCI0_TXI0)  ************************************************/
324
325 #pragma interrupt (Int_SCI0_TXI0(enable))
326 void Int_SCI0_TXI0(void)
327 {
328     /* 送信データの有無チェック */
329     if(tx0_rptr != tx0_size){                   /* 送信データあり  */
330         /* データ送信 */
331         SCI0.TDR = *(tx0_addr + tx0_rptr);
332         /* 送信データポインタ更新 */
333         tx0_rptr++;
334         if(tx0_rptr == tx0_size){
```

```
335             IEN(SCI0, TXI0) = 0;                /* 送信割り込み禁止     */
336             IEN(SCI0, TEI0) = 1;                /* 送信完了割り込み許可 */
337         }
338     }
339 }
340
341 /*  送信終了割り込み(SCI0_TEI0)  ********************************************/
342
343 #pragma interrupt (Int_SCI0_TEI0)
344 void Int_SCI0_TEI0(void)
345 {
346     SCI0_RTS = OFF;                             /* RTS=OFF */
347     IEN(SCI0, TEI0) = 0;                        /* 送信完了割り込み禁止 */
348 }
349
350
351 /*****************************************************************************/
352 /*                          １フレーム送信                                   */
353 /*****************************************************************************/
354 /* 引数 : wTimeOut = 送信待ち時間 (0～65535, ×MCLK0単位)                     */
355 /*      :  pbyData = 送信フレームの先頭アドレス                              */
356 /*      :             +0 : 送信データバイト数n(H)                            */
357 /*      :             +1 : 送信データバイト数n(L)                            */
358 /*      :             +2 : 送信データの先頭                                  */
359 /*      :                :                                                  */
360 /*      :          +n+2 : CRC(L)                                            */
361 /*      :          +n+3 : CRC(H)                                            */
362 /*      :            ※バイト数にはバイト数自身とCRCは含まない。             */
363 /* 戻値 : SendFrame_sci0() ==  1 : 正常                                      */
364 /*      :                  == -1 : 異常、タイムアウト                        */
365
366 int SendFrame_sci0(WORD wTimeOut, const BYTE *pbyData)
367 {
368     /* 送信待ち時間セット */
369     wTmrTxWait = wTimeOut;
370     /* 送信完了していない、または４バイト分以上回線が空いていない場合は待つ */
371     while((sci0_stat.BIT.CHAR4TIME == 0) || (sci0_stat.BIT.TXCOMPL == 0)){
372         if(wTmrTxWait == 0)
373             return -1;
374     }
375     /* 送信データサイズセット */
376     get_word(&tx0_size, pbyData);
377     /* データバイト数＋CRCバイト数加算 */
378     tx0_size += 4;
379     /* 送信データのアドレスセット */
380     tx0_addr = (BYTE*)pbyData;
381     /* 送信パラメータセット */
382     tx0_rptr = 0;                               /* 送信リードポインタ */
383     tx0_cptr = 0;                               /* 送信確認用ポインタ */
384     /* 送信完了フラグクリア */
385     sci0_stat.BIT.TXCOMPL  = 0;
386     sci0_stat.BIT.SENDMODE = 1;
387     /* RTS=ON */
388     SCI0_RTS = ON;
389     /* 送信割り込み許可 */
390     IEN(SCI0, TXI0) = 1;
391
392     return 1;
393 }
394
395 /*****************************************************************************/
396 /*                          １フレーム受信                                   */
397 /*****************************************************************************/
398 /* 引数 : wTimeOut = 受信待ち時間 (0～65535, ×MCLK0単位)                     */
399 /*      :   wRxMax = 受信バッファのサイズ (pbyBuffのサイズ)                  */
400 /*      :  pbyBuff = 受信フレームの格納アドレス (フレーム全体)               */
401 /*      :             +0 : データバイト数n(H)                                */
402 /*      :             +1 : データバイト数n(L)                                */
403 /*      :             +2 : データの先頭                                      */
404 /*      :                :                                                  */
405 /*      :          +n+2 : CRC(L)                                            */
406 /*      :          +n+3 : CRC(H)                                            */
407 /*      :            ※バイト数にはバイト数自身とCRCは含まない。             */
408 /* 戻値 : RecvFrame_sci0() >=  0 : 受信フレーム全体のサイズ (CRCのサイズ含む) */
409 /*      :                  == -1 : タイムアウト                              */
410 /*      :                  == -2 : 受信エラー                                */
411
412 int RecvFrame_sci0(WORD wTimeOut, WORD wRxMax, BYTE *pbyBuff)
413 {
414     BYTE    *pBuff;
415     WORD    wSize,wLpcnt;
416
417     /* 受信待ち時間セット (10mS単位)  */
418     wTmrRxWait = wTimeOut;
```

```
419     /* 受信待ち */
420     while(sci0_stat.BIT.RXCOMPL == 0){
421         if(wTmrRxWait == 0)
422             return -1;
423     }
424     /* データバイト数取得 */
425     get_word(&wSize, rx0_buff);
426     /* バイト数とCRCのバイト数を加算 */
427     wSize += 4;
428     /* 受信フレームをコピー */
429     pBuff = rx0_buff;
430     for(wLpcnt=0;(wLpcnt < wSize) && (wLpcnt < wRxMax);wLpcnt++){
431         *(pbyBuff++) = *(pBuff++);
432     }
433     /* 受信パラメータクリア */
434     rx0_wptr = 0;
435     sci0_stat.BIT.RECEIVED = 0;
436     sci0_stat.BIT.RXCOMPL  = 0;
437 
438     return (int)wSize;
439 }
440 
441 /*****************************************************************************/
442 /*                          ＣＲＣ－ＣＣＩＴＴを計算する                    */
443 /*****************************************************************************/
444 /* ※プロトコルによっては初期値等が異なる場合がある。                        */
445 /* 注意 : マイコンのCRCレジスタは静的変数と同じで再帰に対応できないため     */
446 /*      : 計算中は割り込みマスクする。                                       */
447 /* 構文 : WORD crc_ccitt(WORD wSize, BYTE* pData);                           */
448 /* 引数 : wSize = データのバイト数                                           */
449 /*      : pData = 計算するデータの先頭アドレス                               */
450 /* 戻値 : crc_ccitt() = CRCの値 (CRC-CCITT)                                  */
451 
452 WORD crc_ccitt(WORD wSize, const BYTE* pData)
453 {
454     DWORD   dwPSW;
455     WORD    wCRC;
456 
457     /* 割り込みマスク */
458     dwPSW = get_psw();
459     clrpsw_i();
460     CRC.CRCCR.BYTE = 0x83;          /* 生成多項式(X16+X12+X5+1), LSBファースト指定 */
461     CRC.CRCDOR = 0xFFFF;            /* 初期値=0xFFFF */
462     while(wSize-- != 0){
463         CRC.CRCDIR = *(pData++);
464     }
465     wCRC = (WORD)CRC.CRCDOR;
466     /* 割り込みマスク復帰 */
467     set_psw(dwPSW);
468 
469     return wCRC;
470 }
471 
472 /*****************************************************************************/
473 /***                    コマンド受信－レスポンス送信支援                   ***/
474 /*****************************************************************************/
475 
476 #pragma section
477 static BYTE RcvBuff[RX0BUFFSZ];
478 static BYTE SndBuff[TX0BUFFSZ];
479 static BYTE byRxSequensNum;
480 
481 /*****************************************************************************/
482 /*                            コマンドを受信する                             */
483 /*****************************************************************************/
484 /* 引数 : pbyCmdCode = コマンドコード格納アドレス                            */
485 /*      :     pwSize = コマンドのバイト数格納アドレス                        */
486 /*      :     pParam = コマンド格納アドレス (RX0BUFFSZ-6バイト以上必要)      */
487 /* 戻値 : ReceiveCommand() ==  1 : 正常                                      */
488 /*      :                  ==  0 : CRC異常                                   */
489 /*      :                  == -1 : タイムアウトまたはフレーム異常            */
490 
491 int ReceiveCommand(BYTE *pbyCmdCode, WORD *pwSize, BYTE *pParam)
492 {
493     BYTE    *pRcvBuff;
494     int     nRxSize;
495     WORD    wCRC;
496 
497     /* コマンドを受信 */
498     nRxSize = RecvFrame_sci0(50/MCLK0, RX0BUFFSZ, RcvBuff);
499     if(nRxSize < 6)
500         return -1;                          /* 受信タイムアウトまたはフレーム異常 */
501     /* CRCを計算 */
502     wCRC = crc_ccitt((WORD)nRxSize, RcvBuff);
```

```
503     if(wCRC != 0)
504         return 0;
505     /* シーケンス番号を取り込む */
506     pRcvBuff = &RcvBuff[2];
507     byRxSequensNum = *(pRcvBuff++);
508     /* コマンドコードを取り込む */
509     *pbyCmdCode = *(pRcvBuff++);
510     /* データバイト数とシーケンス番号とコマンドコードとCRCのバイト数を減算 */
511     nRxSize -= 6;
512     *pwSize = (WORD)nRxSize;
513     /* パラメータをコピー */
514     if(nRxSize > 0){
515         copy_data_s((WORD)nRxSize, pParam, pRcvBuff);
516     }
517     return 1;
518 }
519 
520 /*****************************************************************************/
521 /*                          レスポンスを送信する                             */
522 /*****************************************************************************/
523 /* 引数 : byResCode = レスポンスコード (DSP_****)                            */
524 /*      :     wSize = レスポンスのバイト数 (0～TXOBUFFSZ-6)                  */
525 /*      :    pParam = レスポンスの先頭アドレス                               */
526 /* 戻値 : SendResponse() ==  1 : 正常                                        */
527 /*      :                ==  0 : 異常あり (長すぎる)                        */
528 /*      :                == -1 : 送信不可                                   */
529 
530 int SendResponse(BYTE byResCode, WORD wSize, const BYTE *pParam)
531 {
532     BYTE    *pSndBuff;
533     WORD    wCRC;
534 
535     /* 長すぎるものは無効 */
536     if((TXOBUFFSZ - 6) < wSize)
537         return 0;
538     /* バイト数をセット */
539     pSndBuff = put_word((WORD)(wSize+2), SndBuff);  /* シーケンス番号とコマンドコードのバイト数を加算 */
540     /* シーケンス番号をセット */
541     *(pSndBuff++) = byRxSequensNum;
542     /* コマンドコードをセット */
543     *(pSndBuff++) = byResCode;
544     /* パラメータをコピー */
545     if(wSize != 0){
546         pSndBuff = copy_data_d(wSize, pSndBuff, pParam);
547     }
548     /* CRCを計算 */
549     wCRC = crc_ccitt((WORD)(wSize + 4), SndBuff);   /* バイト数とシーケンス番号とコマンドコードのバイト数を加算 */
550     /* CRCを格納 */
551     *(pSndBuff++) = (BYTE)wCRC;
552     *(pSndBuff++) = (BYTE)(wCRC >> 8);
553 
554     /* レスポンスを送信 (100mSまで)  */
555     return SendFrame_sci0(100/MCLK0, SndBuff);
556 }
557 
558 
```