```
/*****************************************************************************/
/*****************************************************************************/
/***                                                                       ***/
/***                          TWI(I2C)通信制御                             ***/
/***                                                    by : S.Suzuki      ***/
/*****************************************************************************/
/*****************************************************************************/
/* 注意 : システムクロック（ここではPCLK）は、ヒューズビットによって 8MHzと    */
/*      : してあること。レジスタCLKPRは初期値のまま変更していない。           */

#include <avr/io.h>
#include <avr/interrupt.h>

#include "hdwareInfo.h"     // ←BYTE,WORD等独自の型宣言あり
#include "TWI_I2C.h"

/*****************************************************************************/
/* 参考 : TWBR = (PCLK / SCL / TWPS - 16) / 2                                */
/*      : PCLK = CPUクロック（20MHz以下）                                    */
/*      :  SCL = I2Cのクロック（400KHz以下）                                 */
/*      : TWPS = 前置分周器による分周比（1,4,16,64のどれか）                */
/*      :        TWSR.TWPS[1-0] = 00:分周比=1                               */
/*      :                       = 01:分周比=4                               */
/*      :                       = 10:分周比=16                              */
/*      :                       = 11:分周比=64                              */
/*      : 上記により                                                         */
/*      :   100Kbpsのとき TWBR = 32                                          */
/*      :   400Kbpsのとき TWBR = 2                                           */
/*      :   ただし TWSR.TWPS[1-0]=00, PCLK=8MHzであること。                  */

#define SCL_HZ  100         /* '100'で100kbps、'400'で400kbps、その他は指定不可 */

/* TWSRが返すステータスフラグ */
#define STR_ACK (0x08)      /* スタートコンディション正常          */
#define RST_ACK (0x10)      /* 再スタートコンディション正常        */
#define IDW_ACK (0x18)      /* スレーブアドレス＋書き込み指定正常 */
#define IDW_NAK (0x20)      /* スレーブアドレス＋書き込み指定異常 */
#define IDR_ACK (0x40)      /* スレーブアドレス＋読み込み指定正常 */
#define IDR_NAK (0x48)      /* スレーブアドレス＋読み込み指定異常 */
#define DWR_ACK (0x28)      /* データ書き込み正常 */
#define DWR_NAK (0x30)      /* データ書き込み異常 */
#define DRD_ACK (0x50)      /* データ読み込み正常 */
#define DRD_NAK (0x58)      /* データ読み込み異常または最後の読み込み */
#define BUS_ERR (0x38)      /* バス競合調停失敗    */
/* 処理プロセスを示すコード（st_byProcessの値） */
#define PROC_WAITNG (0)     /* 待機中 */
#define PROC_START  (1)     /* スタートコンディション発行      */
#define PROC_START2 (2)     /* 再スタートコンディション発行    */
#define PROC_SLVIDW (3)     /* スレーブアドレス＋Ｗを書き込み */
#define PROC_SLVIDR (4)     /* スレーブアドレス＋Ｒを書き込み */
#define PROC_DATA1W (5)     /* データ１を書き込み              */
#define PROC_DATA2W (6)     /* データ２を書き込み              */
#define PROC_DATAR  (7)     /* データを読み込み                */
#define PROC_ERROR  (8)     /* エラー終了                      */
//#define PROC_END  (9)     /* 終了                            */

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

/*****************************************************************************/
/* 注意 : (1)「１ブロック目の送信データ」とは、RTCのレジスタ番号やEEPROMのア  */
/*      :    ドレスなど、あらかじめ読み込み／書き込みするデータの場所を指定す */
/*      :    るもの。                                                        */
/*      : (2)「２ブロック目の送信データ」とは、１ブロック目データで指定した  */
/*      :    場所のデータ本体を示す。                                        */

static const BYTE*  st_pSndDataPtr1;    /* １ブロック目の送信データがあるアドレス */
static const BYTE*  st_pSndDataPtr2;    /* ２ブロック目の送信データがあるアドレス */
static BYTE*    st_pRcvBuffPtr;         /* 受信データを格納するアドレス            */
static BYTE     st_bySndSize1;          /* １ブロック目の送信データのバイト数     */
static BYTE     st_bySndSize2;          /* ２ブロック目の送信データのバイト数     */
static BYTE     st_byRcvSize;           /* 受信データのバイト数 */
static BYTE     st_bySndCount;          /* 送信バイト数カウンタ */
static BYTE     st_byRcvCount;          /* 受信バイト数カウンタ */
static BYTE     st_bySlaveID;           /* スレーブのアドレス   */

static BYTE      st_bySlaveID;          /* 相手（スレーブ）のアドレス */
static volatile BYTE st_byProcess;      /* 処理プロセス番号 */

```

```
/*****************************************************************************/
/*                              定周期割り込み処理                           */
/*****************************************************************************/
/* ※1mS周期割り込みで呼ぶこと。 ←正しく呼ばないと正常に動かないので注意！    */

static volatile WORD wTmrEndWait;

void Timer01mS_I2C(void)
{
    if(wTmrEndWait != 0){
        --wTmrEndWait;
    }
}

/*****************************************************************************/
/*                          TWI(I2C)インターフェース初期化                   */
/*****************************************************************************/
/* 注意 : 割り込みマスク状態で、起動後に一度呼ぶこと。                       */
/* 参考 : TWCR.TWEN=1でPC5,PC4端子はSCL,SDAになる。                          */

void Init_I2C(void)
{
    /* I/Oポートの初期化 (TWCR.TWEN=0としたときPC5,PC4が入力ポートになるよう明示的に設定)  */
    DDRC  &= ~(_BV(DDC5) | _BV(DDC4));       /* PC5,PC4を入力ポートにする */
    PORTC &= ~(_BV(PORTC5) | _BV(PORTC4));  /* PC5,PC4のプルアップOFF     */

    /* TWI(I2C)の転送速度(SCL)を設定 */
#if (SCL_HZ == 100)
    TWBR = 32;
#elif (SCL_HZ == 400)
    TWBR = 2;
#else
    #error 106:define Error 'SCL_HZ'
#endif

    /* 転送速度を決めるプリスケラを設定 */
    TWSR = 0x00;              /* TWPS[1-0]='00'でプリスケラはPCLK/1  */
    /* TWI(I2C)の動作開始 */
    TWCR = _BV(TWEN);

    /* 作業用変数類を初期化 */
    st_bySlaveID = 0x00;
    st_byProcess = PROC_WAITNG;
//  st_byTmr01mS = 0;
}

/*****************************************************************************/
/*                      TWI(I2C)インターフェース割り込み処理                 */
/*****************************************************************************/

static inline void Stop_I2C(void);
static inline void StopCond_I2C(void);

/* スタートコンディション送信完了割り込み(STR_ACK:0x08)  **********************/

static void StartSeq_ACK(void)
{
    /* 書き込みモードの時 */
    if((st_bySndSize1 != 0) || (st_bySndSize2 != 0)){
        /* スレーブアドレス＋W書き込み済みを示す */
        st_byProcess = PROC_SLVIDW;
        /* スレーブアドレス＋Wビット送信 */
        TWDR = (BYTE)(st_bySlaveID & 0xFE);
        /* TWCRのTWINTを解除 */
        TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWIE);  /* 割り込み許可 */
    }
    /* 読み込みモードの時 */
    else if(st_byRcvSize != 0){
        /* スレーブアドレス＋Ｒ書き込み済みを示す */
        st_byProcess = PROC_SLVIDR;
        /* スレーブアドレス＋Ｒビット送信 */
        TWDR = (BYTE)(st_bySlaveID | 0x01);
        /* TWCRのTWINTを解除 */
        TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWIE);  /* 割り込み許可 */
    }
    else{
        /* ストップコンディション発行 */
        Stop_I2C();
```

```
157     }
158 }
159
160 /* スタートコンディション再送信完了割り込み(RST_ACK:0x10)  ********************/
161 /* 注意 : スタートコンディション再送信後は読み込みのみとする。                  */
162
163 static void Start2Seq_ACK(void)
164 {
165     /* スレーブアドレス＋Ｒ書き込み済みを示す */
166     st_byProcess = PROC_SLVIDR;
167     /* スレーブアドレス＋Ｒビット送信 */
168     TWDR = (BYTE)(st_bySlaveID | 0x01);
169     /* TWCRのTWINTを解除 */
170     TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWIE);  /* 割り込み許可 */
171 }
172
173 /* スレーブアドレス＋Ｗ送信の応答受信割り込み(IDW_ACK:0x18,DWR_ACK:0x28) ******/
174 /* 注意 : 書き込みと読み込みが連続する場合、最初に書き込みとする。            */
175
176 static void SendID_Data_ACK(void)
177 {
178     /* １ブロック目の送信データがある場合 */
179     if(st_bySndSize1 != 0){
180         st_bySndSize1--;
181         /* データ１書き込み済みを示す */
182         st_byProcess = PROC_DATA1W;
183         /* データを送信 */
184         TWDR = *(st_pSndDataPtr1++);
185         /* TWCRのTWINTを解除 */
186         TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWIE);  /* 割り込み許可 */
187     }
188     /* ２ブロック目の送信データがある場合 */
189     else if(st_bySndSize2 != 0){
190         st_bySndSize2--;
191         /* データ２書き込み済みを示す */
192         st_byProcess = PROC_DATA2W;
193         /* データを送信 */
194         TWDR = *(st_pSndDataPtr2++);
195         /* TWCRのTWINTを解除 */
196         TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWIE);  /* 割り込み許可 */
197     }
198     /* 引き続き受信データを要求する場合 */
199     else if(st_byRcvSize != 0){
200         /* 処理プロセス番号をセット */
201         st_byProcess = PROC_START2;
202         /* スタートコンディション再発行 */
203         TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN) | _BV(TWIE); /* 割り込み許可 */
204     }
205     /* 送信データも受信データもない場合 */
206     else{
207         /* ストップコンディション発行 */
208         Stop_I2C();
209     }
210 }
211
212 /* スレーブアドレス＋Ｒ送信の応答受信割り込み(IDR_ACK:0x40) ******************/
213
214 static void SendID_R_ACK(void)
215 {
216     /* 処理プロセス番号をセット */
217     st_byProcess = PROC_DATAR;
218     /* 受信バイト数が１バイトの場合 */
219     if(st_byRcvSize == 1){
220         /* NAK発行 */
221         TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWIE);  /* 割り込み許可 */
222     }
223     /* 受信バイト数が複数バイトの場合 */
224     else{
225         /* ACK発行 */
226         TWCR = _BV(TWINT) | _BV(TWEA) | _BV(TWEN) | _BV(TWIE);  /* 割り込み許可 */
227     }
228 }
229
230 /* 読み込みデータ受信の応答受信割り込み(DRD_ACK:0x50) ************************/
231
232 static void ReadData_ACK(void)
233 {
234     /* 処理プロセス番号をセット */
```

```
235 //  st_byProcess = PROC_DATAR;
236     /* データを受信 */
237     *(st_pRcvBuffPtr++) = TWDR;
238     st_byRcvSize--;
239     /* 受信バイト数が１バイトの場合 */
240     if(st_byRcvSize == 1){
241         /* NAK発行 */
242         TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWIE);  /* 割り込み許可 */
243     }
244     /* 受信バイト数が０バイトの場合（念のため） */
245     else if(st_byRcvSize == 0){
246         /* ストップコンディション発行 */
247         Stop_I2C();
248     }
249     else{
250         /* ACK発行 */
251         TWCR = _BV(TWINT) | _BV(TWEA) | _BV(TWEN) | _BV(TWIE);  /* 割り込み許可 */
252     }
253 }
254
255 /* 最後の読み込みデータ受信の応答受信割り込み(DRD_NAK:0x58) ************************/
256
257 static void ReadData_NAK(void)
258 {
259     /* データを受信 */
260     *st_pRcvBuffPtr = TWDR;
261     /* ストップコンディション発行 */
262     Stop_I2C();
263 }
264
265 /* TWI(I2C)割り込み処理 ***************************************************/
266
267 ISR(TWI_vect)
268 {
269     BYTE    byTWIstat;
270
271     byTWIstat = (BYTE)(TWSR & 0xF8);
272     switch(byTWIstat){
273         /* スタートコンディション発行後の正常応答 */
274         case STR_ACK:   // 0x08
275             StartSeq_ACK();
276             break;
277         /* 再スタートコンディション発行後の正常応答 */
278         case RST_ACK:   // 0x10
279             Start2Seq_ACK();
280             break;
281         /* デバイスID＋書き込み指定後の正常応答 */
282         case IDW_ACK:   // 0x18
283             SendID_Data_ACK();
284             break;
285         /* デバイスID＋読み込み指定後の正常応答 */
286         case IDR_ACK:   // 0x40
287             SendID_R_ACK();
288             break;
289         /* １byteデータ書き込み後の正常応答 */
290         case DWR_ACK:   // 0x28
291             SendID_Data_ACK();
292             break;
293         /* １byteデータ読み込み後の正常応答 */
294         case DRD_ACK:   // 0x50
295             ReadData_ACK();
296             break;
297         /* 最後のデータ読み込み後の応答 */
298         case DRD_NAK:   // 0x58
299             ReadData_NAK();
300             break;
301         /* デバイスID＋書き込み指定後の異常応答 */
302 //      case IDW_NAK:   // 0x20
303 //          break;
304         /* デバイスID＋読み込み指定後の異常応答 */
305 //      case IDR_NAK:   // 0x48
306 //          break;
307         /* データ書き込み後の異常応答 */
308 //      case DWR_NAK:   // 0x30
309 //          break;
310         /* バス競合調停失敗   */
311 //      case BUS_ERR:   // 0x38
312 //          break;
```

```
313         /* 未定義のステータス、未定義の異常発生 */
314         default:
315             /* ストップコンディション発行 */
316             StopCond_I2C();
317             s_byProcess = PROC_ERROR;
318             break;
319     }
320 }
321
322 /*****************************************************************************/
323 /*                              RIIC通信スタート                             */
324 /*****************************************************************************/
325
326 static void Start_I2C(void)
327 {
328     /* 送信・受信カウンタをクリア */
329     st_bySndCount = 0;
330     st_byRcvCount = 0;
331     /* 処理プロセス番号をセット */
332     st_byProcess = PROC_START;
333     /* スタートコンディション発行 */
334     TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN) | _BV(TWIE); /* 割り込み許可 */
335 }
336
337
338 /* ストップコンディション発行 ************************************************/
339 /* 備考 : (1)異常状態からの復帰にも使用できる。                               */
340 /*      : (2)SCL,SDA端子はハイ・インピーダンス状態になる                      */
341
342 static inline void Stop_I2C(void)
343 {
344     /* 処理プロセス番号を初期化する */
345     st_byProcess = PROC_WAITNG;
346     /* ストップコンディション発行 */
347     TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); /* 割り込み禁止 */
348 }
349
350 static inline void StopCond_I2C(void)
351 {
352     /* ストップコンディション発行 */
353     TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN); /* 割り込み禁止 */
354 }
355
356 /*****************************************************************************/
357 /*                            スレーブアドレスの設定                         */
358 /*****************************************************************************/
359 /* 引数 : bySlaveAddr = 相手のスレーブアドレス                                */
360 /* 戻値 : なし                                                               */
361
362 void SetSlaveAddr_I2C(BYTE bySlaveAddr)
363 {
364     /* （相手）スレーブアドレス保存 */
365     st_bySlaveID = bySlaveAddr;
366 }
367
368
369 /*****************************************************************************/
370 /*                        送信してレスポンスを受信する                       */
371 /*****************************************************************************/
372 /* 注意 : スレーブアドレスはあらかじめ設定しておき、１バイト目に送信される。 */
373 /* 解説 : 読み／書きモードとリスタートコンディションの有無                   */
374 /* bySndDataSz* | byRcvDataSz |                                              */
375 /* -------------|-------------|----------------|---------------------------- */
376 /*  == 0        |  == 0       | 書き込みのみ   | リスタートコンディションなし */
377 /*  == 0        |  != 0       | 読み込みのみ   | リスタートコンディションなし */
378 /*  != 0        |  == 0       | 書き込みのみ   | リスタートコンディションなし */
379 /*  != 0        |  != 0       | 書き→読み込み | リスタートコンディションあり */
380 /* 引数 : TxRxParam = 送信／受信パラメータのアドレス                          */
381 /* 戻値 : TranceiveData_I2C() == FALSE : エラーあり（タイムアウトは100mS）    */
382 /*      :                     ==  TRUE : 正常                                */
383
384 BOOL TranceiveData_I2C(I2C_TXRX* TxRxParam)
385 {
386     /* 送信パラメータセット */
387     st_pSndDataPtr1 = TxRxParam->pSndData1;
388     st_bySndSize1   = TxRxParam->bySndDataSz1;
389     st_pSndDataPtr2 = TxRxParam->pSndData2;
390     st_bySndSize2   = TxRxParam->bySndDataSz2;
```

```
391     /* 受信パラメータセット */
392     st_pRcvBuffPtr  = TxRxParam->pRcvBuff;
393     st_byRcvSize    = TxRxParam->byRcvDataSz;
394
395
396     /* TWI(I2C)通信スタート */
397     Start_I2C();
398     /* 通信終了を100mSまで待つ */
399     wTmrEndWait = 100;
400     while(st_byProcess != PROC_WAITNG){
401         if(s_byProcess == PROC_ERROR){
402             s_byProcess = PROC_WAITNG;
403             return FALSE;
404         }
405         if(wTmrEndWait == 0){
406             /* TWI(I2C)通信ストップ */
407             Stop_I2C();
408             return FALSE;
409         }
410     }
411
412     return TRUE;
413 }
414
415 /*****************************************************************************/
416 /*****************************************************************************/
417 /***                                                                       ***/
418 /***                          TWI(I2C)通信支援関数                         ***/
419 /***                                                                       ***/
420 /*****************************************************************************/
421 /*****************************************************************************/
422 #define TRYMAX_I2C  (5)     /* リトライを含む最大送信回数 */
423
424 /*****************************************************************************/
425 /*                              １バイトデータリード                         */
426 /*****************************************************************************/
427 /* １バイトのアドレス指定（RTC用） ******************************************/
428 /* 引数 : byRdAddr = アドレス番号                                            */
429 /*      :  pRxBuff = 読み出しデータを格納するバッファのアドレス              */
430 /* 戻値 : ReadByte_A8_I2C() == FALSE : 処理中（ビジー）                      */
431 /*      :                   ==  TRUE : 正常受付                              */
432
433 BOOL ReadByte_A8_I2C(BYTE byRdAddr, BYTE *pRxBuff)
434 {
435     I2C_TXRX RIICParam;
436     int     nTryCntr;
437
438     /* 送受信パラメータセット */
439     RIICParam.bySndDataSz1 = 1;
440     RIICParam.pSndData1    = (const BYTE*)&byRdAddr;
441     RIICParam.bySndDataSz2 = 0;
442     RIICParam.pSndData2    = NULL;
443     RIICParam.byRcvDataSz  = 1;
444     RIICParam.pRcvBuff     = pRxBuff;
445
446     nTryCntr = 0;
447     do{
448         if(TranceiveData_I2C(&RIICParam) != FALSE){
449             return TRUE;
450         }
451     }while(++nTryCntr < TRYMAX_I2C);
452
453     return FALSE;
454 }
455
456 /* ２バイトのアドレス指定（EEPROM用） ***************************************/
457 /* 引数 : wRdAddr = レジスタアドレス番号                                     */
458 /*      : pRxBuff = 読み出しデータを格納するバッファのアドレス               */
459 /* 戻値 : ReadByte_A16_I2C() == FALSE : 処理中（ビジー）                     */
460 /*      :                    ==  TRUE : 正常受付                          */
461
462 BOOL ReadByte_A16_I2C(WORD wRdAddr, BYTE *pRxBuff)
463 {
464     I2C_TXRX RIICParam;
465     int     nTryCntr;
466     BYTE    abyTxBuff[2];
467
468     /* 16bitアドレスを8bitずつにする */
```

```
469     abyTxBuff[0] = (BYTE)(wRdAddr >> 8);
470     abyTxBuff[1] = (BYTE)wRdAddr;
471 
472     /* 送受信パラメータセット */
473     RIICParam.bySndDataSz1 = 2;
474     RIICParam.pSndData1    = (const BYTE*)abyTxBuff;
475     RIICParam.bySndDataSz2 = 0;
476     RIICParam.pSndData2    = NULL;
477     RIICParam.byRcvDataSz  = 1;
478     RIICParam.pRcvBuff     = pRxBuff;
479 
480     /* 読み込みシーケンス */
481     nTryCntr = 0;
482     do{
483         if(TranceiveData_I2C(&RIICParam) != FALSE){
484             return TRUE;
485         }
486     }while(++nTryCntr < TRYMAX_I2C);
487 
488     return FALSE;
489 }
490 
491 /*****************************************************************************/
492 /*                              ページデータリード                           */
493 /*****************************************************************************/
494 /* １バイトのアドレス指定（RTC用） ******************************************/
495 /* 引数 : byRdAddr = レジスタアドレス番号                                    */
496 /*      :   bySize = 読み込むデータのバイト数                                */
497 /*      :  pRxBuff = 読み出しデータを格納するバッファのアドレス               */
498 /* 戻値 : ReadPage_A8_I2C() == FALSE : 処理中（ビジー）                      */
499 /*      :                   ==  TRUE : 正常受付                              */
500 
501 BOOL ReadPage_A8_I2C(BYTE byRdAddr, BYTE bySize, BYTE *pRxBuff)
502 {
503     I2C_TXRX RIICParam;
504     int     nTryCntr;
505 
506     /* 送受信パラメータセット */
507     RIICParam.bySndDataSz1 = 1;
508     RIICParam.pSndData1    = (const BYTE*)&byRdAddr;
509     RIICParam.bySndDataSz2 = 0;
510     RIICParam.pSndData2    = NULL;
511     RIICParam.byRcvDataSz  = bySize;
512     RIICParam.pRcvBuff     = pRxBuff;
513 
514     nTryCntr = 0;
515     do{
516         if(TranceiveData_I2C(&RIICParam) != FALSE){
517             return TRUE;
518         }
519     }while(++nTryCntr < TRYMAX_I2C);
520 
521     return FALSE;
522 }
523 
524 /* ２バイトのアドレス指定（EEPROM用） ***************************************/
525 /* 引数 : wRdAddr = レジスタアドレス番号                                     */
526 /*      :   bySize = 読み込むデータのバイト数                                */
527 /*      :  pRxBuff = 読み出しデータを格納するバッファのアドレス               */
528 /* 戻値 : ReadPage_A16_I2C() == FALSE : 処理中（ビジー）                     */
529 /*      :                    ==  TRUE : 正常受付                             */
530 
531 BOOL ReadPage_A16_I2C(WORD wRdAddr, BYTE bySize, BYTE *pRxBuff)
532 {
533     I2C_TXRX RIICParam;
534     int     nTryCntr;
535     BYTE    abyTxBuff[2];
536 
537     /* 16bitアドレスを8bitずつにする */
538     abyTxBuff[0] = (BYTE)(wRdAddr >> 8);
539     abyTxBuff[1] = (BYTE)wRdAddr;
540 
541     /* 送受信パラメータセット */
542     RIICParam.bySndDataSz1 = 2;
543     RIICParam.pSndData1    = (const BYTE*)abyTxBuff;
544     RIICParam.bySndDataSz2 = 0;
545     RIICParam.pSndData2    = NULL;
546     RIICParam.byRcvDataSz  = bySize;
```

```
547     RIICParam.pRcvBuff     = pRxBuff;
548 
549     /* 読み込みシーケンス */
550     nTryCntr = 0;
551     do{
552         if(TranceiveData_I2C(&RIICParam) != FALSE){
553             return TRUE;
554         }
555     }while(++nTryCntr < TRYMAX_I2C);
556 
557     return FALSE;
558 }
559 
560 /*****************************************************************************/
561 /*                              １バイトデータライト                         */
562 /*****************************************************************************/
563 /* １バイトのアドレス指定（RTC用） ******************************************/
564 /* 引数 : byWrAddr = レジスタアドレス番号                                    */
565 /*      :  pTxData = 書き込むデータのアドレス                                */
566 /* 戻値 : WriteByte_A8_I2C() == FALSE : 処理中（ビジー）                     */
567 /*      :                         ==  TRUE : 正常受付                       */
568 
569 BOOL WriteByte_A8_I2C(BYTE byWrAddr, const BYTE *pTxData)
570 {
571     I2C_TXRX RIICParam;
572     int     nTryCntr;
573 
574     /* 送受信パラメータセット */
575     RIICParam.bySndDataSz1 = 1;
576     RIICParam.pSndData1    = (const BYTE*)&byWrAddr;
577     RIICParam.bySndDataSz2 = 1;
578     RIICParam.pSndData2    = pTxData;
579     RIICParam.byRcvDataSz  = 0;
580     RIICParam.pRcvBuff     = NULL;
581 
582     nTryCntr = 0;
583     do{
584         if(TranceiveData_I2C(&RIICParam) != FALSE){
585             return TRUE;
586         }
587     }while(++nTryCntr < TRYMAX_I2C);
588 
589     return FALSE;
590 }
591 
592 /* ２バイトのアドレス指定（EEPROM用） ***************************************/
593 /* 引数 : wWrAddr = レジスタアドレス番号                                     */
594 /*      : pTxData = 書き込むデータのアドレス                                 */
595 /* 戻値 : WriteByte_A16_I2C() == FALSE : 処理中（ビジー）                    */
596 /*      :                         ==  TRUE : 正常受付                       */
597 
598 BOOL WriteByte_A16_I2C(WORD wWrAddr, const BYTE *pTxData)
599 {
600     I2C_TXRX RIICParam;
601     int     nTryCntr;
602     BYTE    abyTxBuff[2];
603 
604     /* 16bitアドレスを8bitずつにする */
605     abyTxBuff[0] = (BYTE)(wWrAddr >> 8);
606     abyTxBuff[1] = (BYTE)wWrAddr;
607 
608     /* 送受信パラメータセット */
609     RIICParam.bySndDataSz1 = 2;
610     RIICParam.pSndData1    = (const BYTE*)abyTxBuff;
611     RIICParam.bySndDataSz2 = 1;
612     RIICParam.pSndData2    = pTxData;
613     RIICParam.byRcvDataSz  = 0;
614     RIICParam.pRcvBuff     = NULL;
615 
616     /* 書き込みシーケンス開始 */
617     nTryCntr = 0;
618     do{
619         if(TranceiveData_I2C(&RIICParam) != FALSE){
620             return TRUE;
621         }
622     }while(++nTryCntr < TRYMAX_I2C);
623 
624     return FALSE;
```

```
}


/*****************************************************************************/
/*                          ページデータライト                               */
/*****************************************************************************/
/* １バイトのアドレス指定（RTC用） ******************************************/
/* 引数 : byWrAddr = レジスタアドレス番号                                    */
/*      :   bySize = 書き込むデータのバイト数                                */
/*      :  pTxData = 書き込むデータのアドレス                                */
/* 戻値 : WritePage_A8_I2C() == FALSE : 処理中（ビジー）                     */
/*      :                    ==  TRUE : 正常受付                             */

BOOL WritePage_A8_I2C(BYTE byWrAddr, BYTE bySize, const BYTE *pTxData)
{
    I2C_TXRX RIICParam;
    int     nTryCntr;

    /* 送受信パラメータセット */
    RIICParam.bySndDataSz1 = 1;
    RIICParam.pSndData1    = (const BYTE*)&byWrAddr;
    RIICParam.bySndDataSz2 = bySize;
    RIICParam.pSndData2    = pTxData;
    RIICParam.byRcvDataSz  = 0;
    RIICParam.pRcvBuff     = NULL;

    nTryCntr = 0;
    do{
        if(TranceiveData_I2C(&RIICParam) != FALSE){
            return TRUE;
        }
    }while(++nTryCntr < TRYMAX_I2C);

    return FALSE;
}

/* ２バイトのアドレス指定（EEPROM用） ***************************************/
/* 引数 : wWrAddr = レジスタアドレス番号                                     */
/*      :  bySize = 書き込むデータのバイト数                                 */
/*      : pTxData = 書き込むデータのアドレス                                 */
/* 戻値 : WritePage_A16_I2C() == FALSE : 処理中（ビジー）                    */
/*      :                     ==  TRUE : 正常受付                            */

BOOL WritePage_A16_I2C(WORD wWrAddr, BYTE bySize, const BYTE *pTxData)
{
    I2C_TXRX RIICParam;
    int     nTryCntr;
    BYTE    abyTxBuff[2];

    /* 16bitアドレスを8bitずつにする */
    abyTxBuff[0] = (BYTE)(wWrAddr >> 8);
    abyTxBuff[1] = (BYTE)wWrAddr;
    /* 送受信パラメータセット */
    RIICParam.bySndDataSz1 = 2;
    RIICParam.pSndData1    = (const BYTE*)abyTxBuff;
    RIICParam.bySndDataSz2 = bySize;
    RIICParam.pSndData2    = pTxData;
    RIICParam.byRcvDataSz  = 0;
    RIICParam.pRcvBuff     = NULL;

    /* 書き込みシーケンス開始 */
    nTryCntr = 0;
    do{
        if(TranceiveData_I2C(&RIICParam) != FALSE){
            return TRUE;
        }
    }while(++nTryCntr < TRYMAX_I2C);

    return FALSE;
}


```