```
//18111C_Master
// Mc32Gest_RS232.C
// Canevas manipulatio TP2 RS232 SLO2 2016-2017
// Fonctions d'émission et de réception des message
// CHR 20.12.2016 ajout traitement int error
// CHR 22.12.2016 evolution des marquers observation int Usart
// MDS 26.09.2022 Modification pour permettre la communication avec le Xbee et la gestion des donnee reçu

#include <xc.h>
#include <sys/attribs.h>
#include <stdint.h>
#include "system_definitions.h"
// Ajout CHR
#include "app.h"
#include "Mc32gest_RS232.h"
#include <GenericTypeDefs.h>
//Ajout MDS
#include "GesFifoTh32.h"
#include "RF.h"
#include "Mc32Delays.h"
#include "Data_Code.h"


//definition du byte de fin de trame
#define END  0xBB

//definition du byte de debut de trame
#define START  0xAA

uint8_t countCar = 0;           // Compteur du nombre de characters d'un nom
char buffReadName[20] = {' '};// Buffer de reception de l'UART
uint8_t Name_Receive = false;   //Flag Nom recu

typedef union
{
    uint32_t val32;

    struct
    {
        uint8_t msb;
        uint8_t byte1;
        uint8_t byte2;
        uint8_t lsb;
    }
    shl;
}
U_manip32;
typedef struct {
   uint8_t Start;
   U_32 Add_Master;
   U_32 Add_Slave;
   U_32  Broadcast;
   U_32  data;
   uint8_t End;

} StruMess;

// Struct pour émission des messages
StruMess TxMess;
// Struct pour réception des messages
StruMess RxMess;

// Declaration des FIFO pour réception et émission
#define FIFO_RX_SIZE  ( (63 * 7) + 1)  // 63 messages
#define FIFO_TX_SIZE  ( (63 * 7) + 1)  // 63 messages

int8_t fifoRX[FIFO_RX_SIZE];
// Declaration du descripteur du FIFO de réception
S_fifo descrFifoRX;


int8_t fifoTX[FIFO_TX_SIZE];
```

```
73  // Declaration du descripteur du FIFO d'émission
74  S_fifo descrFifoTX;
75
76  bool LINK,Add_ON;
77  uint32_t Add_Master = 0;
78  uint32_t Add_Slave = 0;
79  bool ID_In_List;
80  bool New_student;
81  uint8_t Nb_In_List;
82
83  // Initialisation de la communication sériel
84  // -----------------------------------------
85
86  void InitFifoComm(void)
87  {
88
89     // Initialisation du fifo de réception
90     InitFifo ( &descrFifoRX, FIFO_RX_SIZE, fifoRX, 0 );
91     // Initialisation du fifo d'émission
92     InitFifo ( &descrFifoTX, FIFO_TX_SIZE, fifoTX, 0 );
93
94  } // InitComm
95
96
97  bool GetMessage(U_32 *pAdd_S,U_32 *pAdd_M, U_32 *pDatas)
98  {
99      bool startOk = false;
100
101     RxMess.End = END;
102     int8_t CarLu,i,y,x;
103     uint8_t car ;
104     if(Get_Add_Master)
105     {
106
107        uint8_t* dstArray ;
108
109         SYS_INT_SourceDisable(INT_SOURCE_USART_1_RECEIVE); //désactive int uart rx
110
111         while(GetReadSize(&descrFifoRX) > 0)
112         {
113
114            GetCharFromFifo (&descrFifoRX, (int8_t*)&CarLu);
115            dstArray[i] = CarLu;
116            i++;
117         }
118         Get_Add_Master = false;
119         Add_Master = (uint32_t)dstArray;
120
121         SYS_INT_SourceEnable(INT_SOURCE_USART_1_RECEIVE); //réactive int uart rx
122     }
123     else
124     {
125         //vérifie longueur message et présence start
126         // trame 14 byte min:
127         // 1 byte de start
128         // 4 byte d'adresse de l'expediteur
129         // 4 byte d'addresse de destinataire
130         // 4 byte de donnee
131         // 1 byte de fin
132          while((GetReadSize(&descrFifoRX)) >= 14)
133         {
134             GetCharFromFifo (&descrFifoRX, &CarLu);
135             if (CarLu == (int8_t)START)
136             {
137                 startOk = true;
138
139                 break;
140             }
141         }
142          //Start trouvé, lire message et décoder
143         if (startOk)
144         {
145             //prendre de la fifo les 4 byte de l'expediteur
```

```
146                 GetCharFromFifo (&descrFifoRX, &CarLu);
147                 pAdd_S->U_32_Bytes.msb = CarLu;
148                 GetCharFromFifo (&descrFifoRX, &CarLu);
149                 pAdd_S->U_32_Bytes.byte1  = CarLu;
150                 GetCharFromFifo (&descrFifoRX, &CarLu);
151                 pAdd_S->U_32_Bytes.byte2  = CarLu;
152                 GetCharFromFifo (&descrFifoRX, &CarLu);
153                 pAdd_S->U_32_Bytes.lsb = CarLu;
154 
155                 //prendre de la fifo les 4 byte du destinataire
156                 GetCharFromFifo (&descrFifoRX, &CarLu);
157                 pAdd_M->U_32_Bytes.msb = CarLu;
158                 GetCharFromFifo (&descrFifoRX, &CarLu);
159                 pAdd_M->U_32_Bytes.byte1  = CarLu;
160                 GetCharFromFifo (&descrFifoRX, &CarLu);
161                 pAdd_M->U_32_Bytes.byte2  = CarLu;
162                 GetCharFromFifo (&descrFifoRX, &CarLu);
163                 pAdd_M->U_32_Bytes.lsb = CarLu;
164 
165                 //lecture byte dans fifo
166                 GetCharFromFifo (&descrFifoRX, &CarLu);
167                 //tant que l'on a pas le byte de fin on enregistre les donnees
168                 do
169                 {
170                     //copie des donnee dans un tableau
171                     buffReadName[countCar] = CarLu;
172                     // lire la prochaine valeur du fifo
173                     GetCharFromFifo (&descrFifoRX, &CarLu);
174                     //incrementation dans le tableau
175                     countCar++;
176                 }
177                 while ( CarLu!= '»');
178 
179                 //si les data sont <4 alors c'est une commande
180                 if(countCar <= 4)
181                 {
182                     //memset(&pDatas, 0x00, sizeof(pDatas));
183                     pDatas->U_32_Bytes.msb = buffReadName[0];
184                     pDatas->U_32_Bytes.byte1 = buffReadName[1];
185                     pDatas->U_32_Bytes.byte2 = buffReadName[2];
186                     pDatas->U_32_Bytes.lsb = buffReadName[3];
187                 }
188                 //si > alors c'est un nom d'eleve
189                 else
190                 {
191                     //on verifie la liste d'eleve
192                     for( i = 0; i < Nb_Student_max ; i++ )
193                     {
194                         x=0;
195                         //on verifie les lettre d'un eleve
196                         for( y = 0; y < countCar ; y++ )
197                         {
198                             //on compare si ce sont les meme
199                             if(buffReadName[countCar] == Students_Info[i].StudentName[
                                countCar])
200                             {
201                                 x++;
202                                 if(x == countCar)
203                                 {
204                                     //flag pour indiquer que le nom est deja dans la list
205                                     ID_In_List = true;
206                                     //on indique quel position dans la list
207                                     Nb_In_List = i;
208                                     // on sort de la boucle
209                                     i = Nb_Student_max;
210                                 }
211                             }
212                             else
213                             {
214                                 //flag pour indiquer que le nom n'est pas dans la list
215                                 ID_In_List = false;
216                                 //on sort de la boucle
217                                 y=countCar;
```

```
                        }
                    }
                }
                if(!ID_In_List)
                {
                    //flag de nouveau eleve
                    New_student=true;
                    for( y = 0; y < countCar ; y++ )
                    {
                        //on copie le nom dans la structure
                        Students_Info[Nb_Student_max].StudentName[y] = buffReadName[y
                        ];
                    }
                    //on copie l'adresse dans la structure
                    Students_Info[Nb_Student_max].ID = pAdd_S->val32;
                    //on copie le nombre de lettre dans le nom dans la structure
                    Students_Info[Nb_Student_max].Name_length = countCar;
                    //on augmente le nombre max d eleve dans la list
                    Nb_Student_max++;
                    //on remet le compteur de data a 0
                    countCar = 0;
                }
            }
            //on enleve le flag de reception xbee
            Message_receive_XBEE = false;
        }
    }
    return startOk;
} // GetMessagetMessage


// Fonction d'envoi des message du Xbee
void SendMessage(uint32_t ADD_M,uint32_t ADD_S, uint32_t pDatas)
{
    uint8_t FreeSize,i,n;
    TxMess.Start = START;
    TxMess.Add_Master.val32 = ADD_M;
    TxMess.Add_Slave.val32 = ADD_S;
    TxMess.data.val32 = pDatas;
    TxMess.End = END;
    TxMess.Broadcast.val32 = ADD_BROADCAST;

    //vérifie longueur disponible dans le fifo
    // trame 14 byte min:
    // 1 byte de start
    // 4 byte d'adresse de l'expediteur
    // 4 byte d'addresse de destinataire
    // 4 byte de donnee
    // 1 byte de fin
    if (GetWriteSpace(&descrFifoTX) >= 14)
    {
        // on met le byte de debut de trame dans le fifo
        PutCharInFifo (&descrFifoTX, TxMess.Start);
        //on vérifie si l'on a deja essayer de se LINK
        if(!LINK)
        {
            // on met l'adresse de broadcast dans le fifo
            PutCharInFifo (&descrFifoTX,  TxMess.Broadcast.U_32_Bytes.msb);
            PutCharInFifo (&descrFifoTX,  TxMess.Broadcast.U_32_Bytes.byte1);
            PutCharInFifo (&descrFifoTX,  TxMess.Broadcast.U_32_Bytes.byte2);
            PutCharInFifo (&descrFifoTX,  TxMess.Broadcast.U_32_Bytes.lsb);
            LINK = true;
        }


        // on met l'adresse de l'expediteur dans le fifo
        PutCharInFifo (&descrFifoTX, TxMess.Add_Master.U_32_Bytes.msb);
        PutCharInFifo (&descrFifoTX, TxMess.Add_Master.U_32_Bytes.byte1);
        PutCharInFifo (&descrFifoTX, TxMess.Add_Master.U_32_Bytes.byte2);
        PutCharInFifo (&descrFifoTX, TxMess.Add_Master.U_32_Bytes.lsb);

        //on vérifie si on a une adresse à envoyer
```

```
290             if(Add_ON)
291             {
292                 // on met l'adresse du destinataire dans le fifo
293                 PutCharInFifo (&descrFifoTX, TxMess.Add_Slave.U_32_Bytes.msb);
294                 PutCharInFifo (&descrFifoTX, TxMess.Add_Slave.U_32_Bytes.byte1);
295                 PutCharInFifo (&descrFifoTX, TxMess.Add_Slave.U_32_Bytes.byte2);
296                 PutCharInFifo (&descrFifoTX, TxMess.Add_Slave.U_32_Bytes.lsb);
297             }
298 
299             // on met les datas dans le fifo
300             PutCharInFifo (&descrFifoTX, TxMess.data.U_32_Bytes.msb);
301             PutCharInFifo (&descrFifoTX, TxMess.data.U_32_Bytes.byte1);
302             PutCharInFifo (&descrFifoTX, TxMess.data.U_32_Bytes.byte2);
303             PutCharInFifo (&descrFifoTX, TxMess.data.U_32_Bytes.lsb);
304 
305             // on met le byte de fin de trame dans le fifo
306             PutCharInFifo (&descrFifoTX, TxMess.End);
307       }
308           // On met le flag d'envoye de l'uart Xbee
309           PLIB_INT_SourceEnable(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT);
310   }
311 
312   // !!!!!!!!
313   // Attention ne pas oublier de supprimer la réponse générée dans system_interrupt
314   // !!!!!!!!
315   void __ISR(_UART_1_VECTOR, ipl5AUTO) _IntHandlerDrvUsartInstance0(void)
316   {
317       USART_ERROR UsartStatus;
318       int8_t Carac, TXsize, TxBuffFull;
319       // Is this an Error interrupt ?
320       if ( PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_1_ERROR) &&
321                PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_1_ERROR) ) {
322           /* Clear pending interrupt */
323           PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_1_ERROR);
324           // Traitement de l'erreur à la réception.
325       }
326 
327 
328       // Is this an RX interrupt ?
329       if ( PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_1_RECEIVE) &&
330                PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_1_RECEIVE) ) {
331 
332           // Oui Test si erreur parité ou overrun
333           UsartStatus = PLIB_USART_ErrorsGet(USART_ID_1);
334 
335           if ( (UsartStatus & (USART_ERROR_PARITY |
336               USART_ERROR_FRAMING | USART_ERROR_RECEIVER_OVERRUN)) == 0)
337           {
338               while(PLIB_USART_ReceiverDataIsAvailable(USART_ID_1))
339               {
340                   //Led_QuestToggle();
341                   Carac = PLIB_USART_ReceiverByteReceive(USART_ID_1);
342                   PutCharInFifo(&descrFifoRX, Carac);
343 
344               }
345               APP_UpdateState(APP_STATE_GET_DATA);
346               // buffer is empty, clear interrupt flag
347               PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_1_RECEIVE);
348               //Message_receive_XBEE = true;
349           } else {
350               // Suppression des erreurs
351               // La lecture des erreurs les efface sauf pour overrun
352               if ( (UsartStatus & USART_ERROR_RECEIVER_OVERRUN) ==
                  USART_ERROR_RECEIVER_OVERRUN) {
353                     PLIB_USART_ReceiverOverrunErrorClear(USART_ID_1);
354               }
355           }
356 
357       } // end if RX
358 
359 
360 
361       // Is this an TX interrupt ?
```

```
362         if ( PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT) &&
363                     PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT) ) {
364
365             TXsize = GetReadSize(&descrFifoTX);
366             TxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_1);
367             if ((TXsize > 0)&& (TxBuffFull == false))
368             {
369                 do//Faire la boucle tant que le message n'est pas envoyé
370                 {
371                     //Led_QuestToggle();
372                     //On va chercher les valeurs a envoyer
373                     GetCharFromFifo(&descrFifoTX, &Carac);
374                     PLIB_USART_TransmitterByteSend(USART_ID_1, Carac);
375                     TXsize = GetReadSize(&descrFifoTX);
376                     TxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_1);
377
378                 }while((TXsize > 0)&& (TxBuffFull == false) && (Carac!= '»'));
379
380                 // Clear the TX interrupt Flag (Seulement apres TX)
381                 PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT);
382                 TXsize = GetReadSize(&descrFifoTX);
383                 if (TXsize == 0)
384                 {
385                     //Pour eviter les interruption inutile
386                     PLIB_INT_SourceDisable(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT);
387                 }
388             }
389             else
390             {
391                   // disable TX interrupt (pour éviter une interrupt. inutile si plus
                      rien à transmettre)
392                  PLIB_INT_SourceDisable(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT);
393             }
394         }
395      }
396
397   // void __ISR(_UART_2_VECTOR, ipl5AUTO) _IntHandlerDrvUsartInstance1(void)
398   //{
399   //    USART_ERROR UsartStatus;
400   //    int8_t Carac, TXsize, TxBuffFull;
401   //    // Is this an Error interrupt ?
402   //    if ( PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_2_ERROR) &&
403   //            PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_2_ERROR) ) {
404   //        /* Clear pending interrupt */
405   //        PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_2_ERROR);
406   //        // Traitement de l'erreur à la réception.
407   //    }
408   //
409   //
410   //    // Is this an RX interrupt ?
411   //    if ( PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_2_RECEIVE) &&
412   //            PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_2_RECEIVE) ) {
413   //
414   //        // Oui Test si erreur parité ou overrun
415   //        UsartStatus = PLIB_USART_ErrorsGet(USART_ID_2);
416   //
417   //        if ( (UsartStatus & (USART_ERROR_PARITY |
418   //            USART_ERROR_FRAMING | USART_ERROR_RECEIVER_OVERRUN)) == 0)
419   //        {
420   //            while(PLIB_USART_ReceiverDataIsAvailable(USART_ID_2))
421   //            {
422   //                Carac = PLIB_USART_ReceiverByteReceive(USART_ID_2);
423   //                PutCharInFifo(&descrFifoRX, Carac);
424   //            }
425   //            // buffer is empty, clear interrupt flag
426   //            PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_2_RECEIVE);
427   //             Message_receive_USB = true;
428   //        } else {
429   //            // Suppression des erreurs
430   //            // La lecture des erreurs les efface sauf pour overrun
431   //            if ( (UsartStatus & USART_ERROR_RECEIVER_OVERRUN) ==
    USART_ERROR_RECEIVER_OVERRUN) {
432   //                PLIB_USART_ReceiverOverrunErrorClear(USART_ID_2);
```

```
433 //            }
434 //        }
435 //
436 //    } // end if RX
437 //
438 //
439 //
440 //    // Is this an TX interrupt ?
441 //    if ( PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT) &&
442 //                 PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT) ) {
443 //
444 //        TXsize = GetReadSize(&descrFifoTX);
445 //        TxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_2);
446 //        if ((TXsize > 0)&& (TxBuffFull == false))
447 //        {
448 //            do//Faire la boucle tant que le message n'est pas envoyé
449 //            {
450 //                //On va chercher les valeurs a envoyer
451 //                GetCharFromFifo(&descrFifoTX, &Carac);
452 //                PLIB_USART_TransmitterByteSend(USART_ID_2, Carac);
453 //                TXsize = GetReadSize(&descrFifoTX);
454 //                TxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_2);
455 //
456 //            }while((TXsize > 0)&& (TxBuffFull == false));
457 //
458 //            // Clear the TX interrupt Flag (Seulement apres TX)
459 //            PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT);
460 //            TXsize = GetReadSize(&descrFifoTX);
461 //            if (TXsize == 0)
462 //            {
463 //                //Pour eviter les interruption inutile
464 //                PLIB_INT_SourceDisable(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT);
465 //            }
466 //        }
467 //        else
468 //        {
469 //              // disable TX interrupt (pour éviter une interrupt. inutile si plus
    rien à transmettre)
470 //             PLIB_INT_SourceDisable(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT);
471 //        }
472 //    }
473 // }
```