```c
//18112C_Slave
//
//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 "Retrieve_name.h"
#include "RF.h"
#include "Mc32Delays.h"
#include "Data_Code.h"
#include "GesFifoTh32.h"


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

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


// Structure décrivant le message (version 2016)


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_First;
   uint8_t Start;
   U_32 Add_Master;
   U_32 Add_Slave;
   U_32 Data;
   uint8_t End;
   char Name [20];

} 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
```

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

```
146             }
147         }
148          //Start trouvé, lire message et décoder
149         if (startOk)
150         {
151 
152             //prendre de la fifo les 4 byte de l'expediteur
153             GetCharFromFifo (&descrFifoRX, &CarLu);
154             pAdd_M->U_32_Bytes.msb = CarLu;
155             GetCharFromFifo (&descrFifoRX, &CarLu);
156             pAdd_M->U_32_Bytes.byte1  = CarLu;
157             GetCharFromFifo (&descrFifoRX, &CarLu);
158             pAdd_M->U_32_Bytes.byte2  = CarLu;
159             GetCharFromFifo (&descrFifoRX, &CarLu);
160             pAdd_M->U_32_Bytes.lsb = CarLu;
161 
162             //On verifie si l'adresse de l'expediteur est un broadcast
163             if(pAdd_M->val32 == ADD_BROADCAST)
164             {
165                 //prendre de la fifo les 4 byte de l'expediteur en ecrasant le
                    broadcast
166                 Message_Broadcast = true;
167                 GetCharFromFifo (&descrFifoRX, &CarLu);
168                 pAdd_M->U_32_Bytes.msb = CarLu;
169                 GetCharFromFifo (&descrFifoRX, &CarLu);
170                 pAdd_M->U_32_Bytes.byte1  = CarLu;
171                 GetCharFromFifo (&descrFifoRX, &CarLu);
172                 pAdd_M->U_32_Bytes.byte2  = CarLu;
173                 GetCharFromFifo (&descrFifoRX, &CarLu);
174                 pAdd_M->U_32_Bytes.lsb = CarLu;
175 
176             }
177             else
178             {
179                 //prendre de la fifo les 4 byte du destinataire
180                 GetCharFromFifo (&descrFifoRX, &CarLu);
181                 pAdd_S->U_32_Bytes.msb = CarLu;
182                 GetCharFromFifo (&descrFifoRX, &CarLu);
183                 pAdd_S->U_32_Bytes.byte1  = CarLu;
184                 GetCharFromFifo (&descrFifoRX, &CarLu);
185                 pAdd_S->U_32_Bytes.byte2  = CarLu;
186                 GetCharFromFifo (&descrFifoRX, &CarLu);
187                 pAdd_S->U_32_Bytes.lsb = CarLu;
188             }
189 
190 
191             //prendre de la fifo les 4 byte de datas
192             GetCharFromFifo (&descrFifoRX, &CarLu);
193             pDatas->U_32_Bytes.msb = CarLu;
194             GetCharFromFifo (&descrFifoRX, &CarLu);
195             pDatas->U_32_Bytes.byte1  = CarLu;
196             GetCharFromFifo (&descrFifoRX, &CarLu);
197             pDatas->U_32_Bytes.byte2  = CarLu;
198             GetCharFromFifo (&descrFifoRX, &CarLu);
199             pDatas->U_32_Bytes.lsb = CarLu;
200 
201             //prendre de la fifo le byte de fin
202             GetCharFromFifo (&descrFifoRX, &CarLu);
203 
204             //on vérifie si on a bien eu le byte de fin
205             if (CarLu == RxMess.End)
206             {
207                 startOk = true;
208             }
209             else
210             {
211                 startOk = false;
212             }
213             //si les data etait un are_you_link
214             if(pDatas->val32 == ARE_U_LINK)
215             {
216                 //Cangement d etat
217                 APP_UpdateState(APP_WAIT_FOR_LINK);
```

```
            }
        }
    }
    return startOk;
} // GetMessagetMessage



// Fonction d'envoi des message
void SendMessage(uint32_t Add_S,uint32_t Add_M, uint32_t Datas)
{
    static uint8_t First_Transmit = true;
    int8_t FreeSize,i;
    // Gestion du control de flux
    TxMess.Start = 0xAA;
    TxMess.End = 0xBB;
    TxMess.Add_Master.val32 = Add_M;
    TxMess.Add_Slave.val32 = Add_S;
    TxMess.Data.val32 = Datas;
    //TxMess.Name = buffReadName;

    //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
    // 4-20 byte de donnee (depend du nom de l'utilisateur)
    // 1 byte de fin
    if (GetWriteSpace(&descrFifoTX) >= (10 + countCar))
    {
        // on met le byte de debut de trame dans le fifo
        PutCharInFifo (&descrFifoTX, TxMess.Start);
        // on met l'adresse de l'expediteur dans le fifo
        PutCharInFifo (&descrFifoTX, TxMess.Add_Slave.U_32_Bytes.msb);
        PutCharInFifo (&descrFifoTX, TxMess.Add_Slave.U_32_Bytes.byte1);
        PutCharInFifo (&descrFifoTX, TxMess.Add_Slave.U_32_Bytes.byte2);
        PutCharInFifo (&descrFifoTX, TxMess.Add_Slave.U_32_Bytes.lsb);

        // on met l'adresse du destinataire 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 c'est le premier envoie de donnee
        if(First_Transmit)
        {
            for(i = 0 ; i <= countCar - 1 ; i ++ )
            {
                //on met les different byte dans le fifo
                PutCharInFifo (&descrFifoTX, buffReadName[i]);

            }
            //on enleve le flag de premiere envoie
            First_Transmit = false;
        }
        else
        {
            // on met les datas dans le fifo
            PutCharInFifo (&descrFifoTX, TxMess.Data.U_32_Bytes.msb);
            PutCharInFifo (&descrFifoTX, TxMess.Data.U_32_Bytes.byte1);
            PutCharInFifo (&descrFifoTX, TxMess.Data.U_32_Bytes.byte2);
            PutCharInFifo (&descrFifoTX, TxMess.Data.U_32_Bytes.lsb);
        }
        PutCharInFifo (&descrFifoTX, TxMess.End);
    }

    PLIB_INT_SourceEnable(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT);
}

// !!!!!!!!
// Attention ne pas oublier de supprimer la réponse générée dans system_interrupt
// !!!!!!!!
```

```
291
292 void __ISR(_UART_1_VECTOR, ipl5AUTO) _IntHandlerDrvUsartInstance0(void)
293 {
294     USART_ERROR UsartStatus;
295     int8_t Carac, TXsize, TxBuffFull;
296     // Is this an Error interrupt ?
297     if ( PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_1_ERROR) &&
298             PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_1_ERROR) ) {
299         /* Clear pending interrupt */
300         PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_1_ERROR);
301         // Traitement de l'erreur à la réception.
302     }
303
304
305     // Is this an RX interrupt ?
306     if ( PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_1_RECEIVE) &&
307             PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_1_RECEIVE) ) {
308
309         // Oui Test si erreur parité ou overrun
310         UsartStatus = PLIB_USART_ErrorsGet(USART_ID_1);
311         if ( (UsartStatus & (USART_ERROR_PARITY |
312             USART_ERROR_FRAMING | USART_ERROR_RECEIVER_OVERRUN)) == 0)
313         {
314
315             while(PLIB_USART_ReceiverDataIsAvailable(USART_ID_1))
316             {
317                 //Led_ReadyToggle();
318                 Carac = PLIB_USART_ReceiverByteReceive(USART_ID_1);
319                 PutCharInFifo(&descrFifoRX, Carac);
320             }
321             //Message_receive = true;
322             // buffer is empty, clear interrupt flag
323             PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_1_RECEIVE);
324         } else {
325             // Suppression des erreurs
326             // La lecture des erreurs les efface sauf pour overrun
327             if ( (UsartStatus & USART_ERROR_RECEIVER_OVERRUN) ==
                USART_ERROR_RECEIVER_OVERRUN) {
328                   PLIB_USART_ReceiverOverrunErrorClear(USART_ID_1);
329             }
330         }
331         //Led_ReadyOff();
332     } // end if RX
333
334
335
336     // Is this an TX interrupt ?
337     if ( PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT) &&
338                  PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT) ) {
339
340         TXsize = GetReadSize(&descrFifoTX);
341         TxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_1);
342         if ((TXsize > 0)&& (TxBuffFull == false))
343         {
344             do//Faire la boucle tant que le message n'est pas envoyé
345             {
346                 //Led_SendedToggle();
347                 //On va chercher les valeurs a envoyer
348                 GetCharFromFifo(&descrFifoTX, &Carac);
349                 PLIB_USART_TransmitterByteSend(USART_ID_1, Carac);
350                 TXsize = GetReadSize(&descrFifoTX);
351                 TxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_1);
352
353             }while((TXsize > 0)&& (TxBuffFull == false));
354
355             // Clear the TX interrupt Flag (Seulement apres TX)
356             PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT);
357             TXsize = GetReadSize(&descrFifoTX);
358             if (TXsize == 0)
359             {
360                 //Pour eviter les interruption inutile
361                 PLIB_INT_SourceDisable(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT);
362             }
```

```
363             }
364             else
365             {
366                   // disable TX interrupt (pour éviter une interrupt. inutile si plus
                      rien à transmettre)
367                   PLIB_INT_SourceDisable(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT);
368             }
369       }
370   }
371
372
373
374
```