

```

1 //18112C_Slave
2 //-----
3 // App.c
4 //-----
5 //
6 //
7 // Auteur      :
8 // Date        :
9 // Version     :
10 // Modifications : MDS 26.09.2022
11 // Description  :
12 //             Application principal de la carte ticketing Master 1811C
13 //
14 //
15 /*-----*/
16 #include "app.h"
17 #include "GesFifoTh32.h"
18 #include "Mc32gest_RS232.h"
19 #include "Retrieve_name.h"
20 #include "Data_Code.h"
21 #include "Mc32Delays.h"
22
23
24
25
26
27 // *****
28 /* Application Data
29
30     Summary:
31         Holds application data
32
33     Description:
34         This structure holds the application's data.
35
36     Remarks:
37         This structure should be initialized by the APP_Initialize function.
38
39         Application strings and buffers are be defined outside this structure.
40 */
41
42 APP_DATA appData;
43 APP_DATA appData_Old;
44
45
46 bool Btn_tickets = true, Btn_tickets_ON;           //Bouton tickets
47 bool flagTickPressed = false;
48 //uint32_t Name_Student = 0x4D6172696F2044;
49
50 // *****
51 // *****
52 // Section: Application Local Functions
53 // *****
54 // *****
55
56
57 /* TODO: Add any necessary local functions.
58 */
59
60
61 // *****
62 // *****
63 // Section: Application Initialization and State Machine Functions
64 // *****
65 // *****
66
67 /*****
68     Function:
69         void APP_Initialize ( void )
70
71     Remarks:
72         See prototype in app.h.
73 */

```

```

74
75 void APP_Initialize ( void )
76 {
77     /* Place the App state machine in its initial state. */
78     appData.state = APP_STATE_INIT;
79
80
81     /* TODO: Initialize your application's state machine and other
82      * parameters.
83      */
84 }
85 void APP_UpdateState (APP_STATES NewState)
86 {
87     appData.state = NewState;
88 }
89
90 /*****
91  Function:
92     void APP_Tasks ( void )
93
94  Remarks:
95     See prototype in app.h.
96  */
97
98 void APP_Tasks ( void )
99 {
100     static int Count = 0;
101     int32_t RXSize;
102     char trash;
103     static uint32_t DataCodeToSend = 0;
104     static bool Ticket_Refused = false;
105     U_32 RXData;
106     U_32 ADD_M;
107     U_32 ADD_S;
108
109
110     /* Check the application's current state. */
111     switch ( appData.state )
112     {
113         /* Application's initial state. */
114         case APP_STATE_INIT:
115         {
116
117             RF_Init();
118             InitFifoComm();
119             //start du timer
120             DRV_TMR0_Start();
121             ALL_LED_OFF();
122             //ALL_LED_ON;
123
124
125             //APP_UpdateState(APP_RETRIEVE_NAME);
126             //appData.state = APP_SEND;
127             appData.state = APP_WAIT_FOR_LINK;
128             //appData.state = APP_READY_TO_SEND;
129
130
131             break;
132         }
133         case APP_RETRIEVE_NAME:
134         {
135
136             ALL_LED_ON();
137             Retrive_Name();
138             if(Name_Receive == true)
139                 APP_UpdateState(APP_WAIT_FOR_LINK);
140                 //Name_Student = atoi( buffReadName);
141                 ALL_LED_OFF();
142             break;
143         }
144         case APP_WAIT_FOR_LINK:
145         {
146

```

```

147 //on récupère le message
148 GetMessage(&ADD_M,&ADD_S,&RXData);
149
150 //on vérifie que ca soit bien le maitre qui nous parle
151 if(Message_Broadcast)
152 {
153     Add_Master = ADD_M.val32;
154     Message_Broadcast = false;
155     //On vérifie que le message reçu est bien un message de link
156     if(RXData.val32 == ARE_U_LINK)
157     {
158         APP_UpdateState(APP_SEND_ID);
159     }
160     else
161     {
162         APP_UpdateState(APP_ERROR);
163     }
164 }
165 else
166 {
167     APP_UpdateState(APP_ERROR);
168 }
169
170 break;
171 }
172 case APP_SEND_ID:
173 {
174     //on prépare le message de réponse
175     //DataCodeToSend = Name_Student;
176     //stop le timer de clignotement des LEDs
177     DRV_TMR0_Stop();
178     ALL_LED_OFF();
179
180     //envoi du message et de l'adresse du module maitre par UART
181     SendMessage(Add_Slave, Add_Master, DataCodeToSend);
182
183     APP_UpdateState(APP_WAIT_FOR_ACK);
184
185
186     appData_Old.state = APP_SEND_ID;
187
188     break;
189 }
190
191 case APP_WAIT_FOR_ACK:
192 {
193
194     //reception du message et de la source
195     GetMessage(&ADD_M,&ADD_S,&RXData);
196
197     //on check que la source est bien le maitre
198     if(ADD_M.val32 == Add_Master)
199     {
200         if(ADD_S.val32 == Add_Slave)
201         {
202             if(RXData.val32 == ACK)
203             {
204                 //comme il sagissait d'un envoi de donné
205                 //on regarde quel etat l'as provoqué pour
206                 //ensuite le rediriger au bon état suivant
207                 switch(appData_Old.state)
208                 {
209                     ALL_LED_ON();
210                     case APP_SEND_ID:
211                     {
212                         ALL_LED_OFF();
213                         DRV_TMR0_Stop();
214                         APP_UpdateState(APP_WAIT_FOR_TICKET);
215                         appData_Old.state = APP_WAIT_FOR_ACK;
216                         break;
217                     }
218                     case APP_READY_TO_SEND:
219                     {

```

```

220 // LED_WAIT_OFF;
221 // appData_Old.state = APP_WAIT_FOR_ACK;
222 // appData.state = APP_WAIT_FOR_TICKET_ACCEPT;
223 // break;
224 // }
225 // case APP_WAIT_FOR_TICKET_ACCEPT:
226 // {
227 // LED_WAIT_OFF;
228 // appData.state = APP_READY_TO_SEND;
229 // break;
230 // }
231 }
232 }
233 }
234 // else
235 // {
236 // APP_UpdateState(APP_ERROR);
237 // }
238 }
239
240 break;
241 }
242
243 case APP_WAIT_FOR_TICKET:
244 {
245 //On allume la LED verte
246 Led_ReadyOn();
247 if(appButtons.Btn_Tickets)
248 {
249 //on prépare l'envoi du ticket
250 DataCodeToSend = ENVOI_TICKET;
251 //LED verte éteinte
252 ALL_LED_OFF();
253 //LED orange allumée
254 Led_SendedOn();
255 APP_UpdateState(APP_SEND_DATA);
256 appData_Old.state = APP_WAIT_FOR_TICKET;
257 }
258
259
260 RXSize = GetReadSize(&descrFifoRX);
261 if(RXSize >= 8)
262 {
263 //reception du message et de la source
264 GetMessage(&ADD_M,&ADD_S,&RXData);
265 //On attend que l'utilisateur appuie sur le bouton
266 if(ADD_M.val32 == Add_Master)
267 {
268 if(ADD_S.val32 == Add_Slave)
269 {
270 if(RXData.val32 == ARE_U_LINK)
271 {
272 APP_UpdateState(APP_SEND_ID);
273 appData_Old.state = APP_WAIT_FOR_TICKET;
274 }
275 }
276 }
277 }
278
279 }
280 break;
281
282
283 }
284
285
286
287
288
289
290
291
292

```

```

293
294
295 case APP_SEND_DATA:
296 {
297     //envoi du message et de l'adresse du module maitre par UART
298     SendMessage(Add_Slave, Add_Master, DataCodeToSend);
299     APP_UpdateState(APP_WAIT_FOR_TICKET_ACCEPT);
300     appData_Old.state = APP_SEND_DATA;
301
302     break;
303 }
304
305
306 case APP_WAIT_FOR_TICKET_ACCEPT:
307 {
308     //Led d'attente
309     Led_SendedOn();
310     //on regarde si on recois un message via l'UART
311
312     //reception du message
313     GetMessage(&ADD_M,&ADD_S,&RXData);
314     //check si la source est bien le maitre
315     if(ADD_M.val32 == Add_Master)
316     {
317         if(ADD_S.val32 == Add_Slave)
318         {
319             //ici on regarde l'info qui nous a été retourné
320             //et selon la réponse retournée et redirige sur les
321             //différents états
322             if(RXData.val32 == TICKET_ACCEPT)
323             {
324                 APP_UpdateState(APP_ACCEPT);
325             }
326             else if(RXData.val32 == TICKET_REFUSE)
327             {
328                 Ticket_Refused = true;
329                 APP_UpdateState(APP_REFUSED);
330             }
331             else if(RXData.val32 == BLOCKED)
332             {
333                 APP_UpdateState(APP_BLOCKED);
334             }
335             else if(RXData.val32 == TICKET_RESET)
336             {
337                 APP_UpdateState(APP_RESET);
338             }
339             else
340             {
341                 APP_UpdateState(APP_ERROR);
342             }
343         }
344     }
345
346     //si l'utilisateur appuie longtemps sur le bouton
347     //il générera une annulation du ticket
348     if(PLIB_PORTS_PinGet(PORTS_ID_0,PORT_CHANNEL_B,PORTS_BIT_POS_7))
349     {
350         Count ++;
351         if(Count >= 5000)
352         {
353             DataCodeToSend = TICKET_ANNULER;
354             APP_UpdateState(APP_SEND_DATA);
355             appData_Old.state = APP_WAIT_FOR_TICKET_ACCEPT;
356         }
357     }
358     else
359     {
360         Count = 0;
361     }
362     break;
363 }
364
365

```

```

366
367
368
369
370
371
372
373
374     case APP_ACCEPT:
375     {
376         //si le ticket a accepté
377         //on fait clignoté la led verte
378         //et on retourne dans le ready to send
379         Led_SendedOff();
380         Blink_LED_ACC();
381         APP_UpdateState(APP_WAIT_FOR_TICKET);
382         break;
383     }
384
385     case APP_REFUSED:
386     {
387         //si le ticket a été refusé
388         //on allume la led rouge
389         //et on bloque l'envoi de ticket pendant un moment
390         //débloquage via le temps ou le reset de ticket
391         Led_Link_LostOn();
392
393         DRV_TMR1_Start();
394         if(Ticket_Refused == false) //débloqué par le timer
395         {
396             DRV_TMR1_Stop();
397             Led_Link_LostOff();
398             APP_UpdateState(APP_WAIT_FOR_TICKET);
399         }
400         //Reception du message de reset
401         RXSize = GetReadSize(&descrFifoRX);
402         if(RXSize >= 8)
403         {
404             GetMessage(&ADD_M,&ADD_S,&RXData);
405             if(ADD_M.val32 == Add_Master)
406             {
407                 if(RXData.val32 == TICKET_RESET)
408                 {
409                     Led_Link_LostOff();
410                     APP_UpdateState(APP_WAIT_FOR_TICKET);
411                 }
412             }
413         }
414         break;
415     }
416
417
418     case APP_ERROR:
419     {
420         //vide le FIFO
421         RXSize = GetReadSize(&descrFifoRX);
422         {
423             while (RXSize > 0)
424             {
425                 GetCharFromFifo(&descrFifoRX, &trash);
426                 RXSize--;
427             }
428             APP_UpdateState(APP_WAIT_FOR_LINK);
429         }
430         break;
431     }
432
433     case APP_RESET:
434     {
435         //Vide le FIFO
436         RXSize = GetReadSize(&descrFifoRX);
437         {
438             while (RXSize > 0)

```

```

439         {
440             GetCharFromFifo(&descrFifoRX, &trash);
441             RXSize --;
442         }
443         APP_UpdateState(APP_WAIT_FOR_TICKET);
444     }
445     break;
446 }
447
448 /* TODO: implement your application state machine.*/
449
450
451 /* The default state should never be executed. */
452 default:
453 {
454     /* TODO: Handle error in application's state machine. */
455     break;
456 }
457 }
458 }
459
460
461 void __ISR(_CHANGE_NOTICE_VECTOR, ipl3AUTO) _IntHandlerChangeNotification(void)
462 {
463
464     //Bouton DECLINE
465     if (PLIB_PORTS_PinGet (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_7))
466     {
467         flagTickPressed = true;
468     }
469     if (flagTickPressed)
470     {
471         if(!PLIB_PORTS_PinGet (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_7))
472         {
473             appButtons.Btn_Tickets = true;
474             flagTickPressed = false;
475         }
476     }
477
478     PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_CHANGE_NOTICE_B);
479 }
480
481 void Blink_LED_ACC (void)
482 {
483     int I;
484     ALL_LED_OFF;
485     for(I = 0; I < 5000; I ++)
486     {
487
488     }
489     ALL_LED_ON;
490     for(I = 0; I < 5000; I ++)
491     {
492
493     }
494     ALL_LED_OFF;
495 }
496
497 void ALL_LED_ON ()
498 {
499     Led_ReadyOn();
500     Led_SendedOn();
501     Led_Link_LostOn();
502 }
503 void ALL_LED_OFF ()
504 {
505     Led_ReadyOff();
506     Led_SendedOff();
507     Led_Link_LostOff();
508 }
509
510 /*****
511 End of File

```

