```c
//18112C_Slave
//---------------------------------------------------------
// App.c
//---------------------------------------------------------
//
//
//  Auteur      :
//  Date        :
//  Version     :
//  Modifications : MDS 26.09.2022
//  Description :
//          Application principal de la carte ticketing Master 1811C
//
//
/*---------------------------------------------------------*/
#include "app.h"
#include "GesFifoTh32.h"
#include "Mc32gest_RS232.h"
#include "Retrieve_name.h"
#include "Data_Code.h"
#include "Mc32Delays.h"




// *****************************************************************************
/* Application Data

  Summary:
    Holds application data

  Description:
    This structure holds the application's data.

  Remarks:
    This structure should be initialized by the APP_Initialize function.

    Application strings and buffers are be defined outside this structure.
*/

APP_DATA appData;
APP_DATA appData_Old;


bool Btn_tickets = true,Btn_tickets_ON;      //Bouton tickets
bool flagTickPressed = false;
//uint32_t Name_Student = 0x4D6172696F2044;

// *****************************************************************************
// *****************************************************************************
// Section: Application Local Functions
// *****************************************************************************
// *****************************************************************************


/* TODO:  Add any necessary local functions.
*/


// *****************************************************************************
// *****************************************************************************
// Section: Application Initialization and State Machine Functions
// *****************************************************************************
// *****************************************************************************

/*******************************************************************************
  Function:
    void APP_Initialize ( void )

  Remarks:
    See prototype in app.h.
 */
```

```c

void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;


    /* TODO: Initialize your application's state machine and other
     * parameters.
     */
}
void APP_UpdateState (APP_STATES NewState)
{
    appData.state = NewState;
}

/******************************************************************************
  Function:
    void APP_Tasks ( void )

  Remarks:
    See prototype in app.h.
 */

void APP_Tasks ( void )
{
    static int Count = 0;
    int32_t RXSize;
    char trash;
    static uint32_t DataCodeToSend = 0;
    static bool Ticket_Refused = false;
    U_32 RXData;
    U_32 ADD_M;
    U_32 ADD_S;


    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_INIT:
        {

            RF_Init();
            InitFifoComm();
            //start du timer
            DRV_TMR0_Start();
            ALL_LED_OFF();
            //ALL_LED_ON;


            //APP_UpdateState(APP_RETRIEVE_NAME);
            //appData.state = APP_SEND;
            appData.state = APP_WAIT_FOR_LINK;
            //appData.state = APP_READY_TO_SEND;


            break;
        }
        case APP_RETRIEVE_NAME:
        {

            ALL_LED_ON();
            Retrive_Name();
            if(Name_Receive == true)
                APP_UpdateState(APP_WAIT_FOR_LINK);
                //Name_Student = atoi( buffReadName);
                ALL_LED_OFF();
            break;
        }
        case APP_WAIT_FOR_LINK:
        {

            //on récupère le message
            GetMessage(&ADD_M,&ADD_S,&RXData);

            //on vérifie que ca soit bien le maitre qui nous parle
            if(Message_Broadcast)
            {
                Add_Master = ADD_M.val32;
                Message_Broadcast = false;
                //On verifie que le message recu est bien un message de link
                if(RXData.val32 == ARE_U_LINK)
                {
                    APP_UpdateState(APP_SEND_ID);
                }
                else
                {
                    APP_UpdateState(APP_ERROR);
                }
            }
            else
            {
                APP_UpdateState(APP_ERROR);
            }

            break;
        }
         case APP_SEND_ID:
        {
            //on prépare le message de réponse
           //DataCodeToSend = Name_Student;
            //stop le timer de clignotement des LEDs
            DRV_TMR0_Stop();
            ALL_LED_OFF();

            //envoi du message et de l'adresse du module maitre par UART
            SendMessage(Add_Slave, Add_Master, DataCodeToSend);

            APP_UpdateState(APP_WAIT_FOR_ACK);


            appData_Old.state = APP_SEND_ID;

            break;
        }

        case APP_WAIT_FOR_ACK:
        {

           //reception du message et de la source
           GetMessage(&ADD_M,&ADD_S,&RXData);

           //on check que la source est bien le maitre
           if(ADD_M.val32 == Add_Master)
            {
               if(ADD_S.val32 == Add_Slave)
                {
                    if(RXData.val32 == ACK)
                    {
                        //comme il sagissait d'un envoi de donné
                        //on regarde quel etat l'as provoqué pour
                        //ensuite le rediriger au bon état suivant
                        switch(appData_Old.state)
                        {
                            ALL_LED_ON();
                            case APP_SEND_ID:
                            {
                                ALL_LED_OFF();
                                DRV_TMR0_Stop();
                                APP_UpdateState(APP_WAIT_FOR_TICKET);
                                appData_Old.state = APP_WAIT_FOR_ACK;
                                break;
                            }
//                                case APP_READY_TO_SEND:
//                                {
```

```
//                                  LED_WAIT_OFF;
//                                  appData_Old.state = APP_WAIT_FOR_ACK;
//                                  appData.state = APP_WAIT_FOR_TICKET_ACCEPT;
//                                  break;
//                              }
//                              case APP_WAIT_FOR_TICKET_ACCEPT:
//                              {
//                                  LED_WAIT_OFF;
//                                  appData.state = APP_READY_TO_SEND;
//                                  break;
//                              }
                        }
                    }
                }
//                else
//                {
//                    APP_UpdateState(APP_ERROR);
//                }
            }

            break;
        }

        case APP_WAIT_FOR_TICKET:
        {
            //On allume la LED verte
            Led_ReadyOn();
            if(appButtons.Btn_Tickets)
            {
                //on prépare l'envoi du ticket
                DataCodeToSend = ENVOI_TICKET;
                //LED verte éteinte
                ALL_LED_OFF();
                //LED orange allumée
                Led_SendedOn();
                APP_UpdateState(APP_SEND_DATA);
                appData_Old.state = APP_WAIT_FOR_TICKET;
            }


            RXSize = GetReadSize(&descrFifoRX);
            if(RXSize >= 8)
            {
               //reception du message et de la source
               GetMessage(&ADD_M,&ADD_S,&RXData);
                //On attend que l'utilisateur appuie sur le bouton
               if(ADD_M.val32 == Add_Master)
               {
                    if(ADD_S.val32 == Add_Slave)
                    {
                        if(RXData.val32 == ARE_U_LINK)
                        {
                            APP_UpdateState(APP_SEND_ID);
                            appData_Old.state = APP_WAIT_FOR_TICKET;
                        }

                    }
               }

            }
            break;


        }
```

```c


        case APP_SEND_DATA:
        {
            //envoi du message et de l'adresse du module maitre par UART
            SendMessage(Add_Slave, Add_Master, DataCodeToSend);
            APP_UpdateState(APP_WAIT_FOR_TICKET_ACCEPT);
            appData_Old.state = APP_SEND_DATA;

            break;
        }


        case APP_WAIT_FOR_TICKET_ACCEPT:
        {
            //Led d'attente
            Led_SendedOn();
            //on regarde si on recois un message via l'UART

            //reception du message
            GetMessage(&ADD_M,&ADD_S,&RXData);
            //check si la source est bien le maitre
            if(ADD_M.val32 == Add_Master)
            {
                if(ADD_S.val32 == Add_Slave)
                {
                    //ici on regarde l'info qui nous a été retourné
                    //et selon la réponse retournée et redirige sur les
                    //diffèrents états
                    if(RXData.val32 == TICKET_ACCEPT)
                    {
                        APP_UpdateState(APP_ACCEPT);
                    }
                    else if(RXData.val32 == TICKET_REFUSE)
                    {
                        Ticket_Refused = true;
                        APP_UpdateState(APP_REFUSED);
                    }
                    else if(RXData.val32 == BLOCKED)
                    {
                        APP_UpdateState(APP_BLOCKED);
                    }
                    else if(RXData.val32 == TICKET_RESET)
                    {
                        APP_UpdateState(APP_RESET);
                    }
                    else
                    {
                        APP_UpdateState(APP_ERROR);
                    }
                }

            }
            //si l'utilisateur appuie longtemps sur le bouton
            //il générera une annulation du ticket
            if(PLIB_PORTS_PinGet(PORTS_ID_0,PORT_CHANNEL_B,PORTS_BIT_POS_7))
            {
                Count ++;
                if(Count >= 5000)
                {
                    DataCodeToSend = TICKET_ANNULER;
                    APP_UpdateState(APP_SEND_DATA);
                    appData_Old.state = APP_WAIT_FOR_TICKET_ACCEPT;
                }
            }
            else
            {
                Count = 0;
            }
            break;
        }


        case APP_ACCEPT:
        {
            //si le ticket a accepté
            //on fait clignoté la led verte
            //et on retourne dans le ready to send
            Led_SendedOff();
            Blink_LED_ACC();
            APP_UpdateState(APP_WAIT_FOR_TICKET);
            break;
        }

        case APP_REFUSED:
        {
            //si le ticket a été refusé
            //on allume la led rouge
            //et on bloque l'envoi de ticket pendant un moment
            //débloquage via le temps ou le reset de ticket
            Led_Link_LostOn();

            DRV_TMR1_Start();
            if(Ticket_Refused == false) //débloqué par le timer
            {
               DRV_TMR1_Stop();
               Led_Link_LostOff();
               APP_UpdateState(APP_WAIT_FOR_TICKET);
            }
            //Reception du message de reset
            RXSize = GetReadSize(&descrFifoRX);
            if(RXSize >= 8)
            {
                GetMessage(&ADD_M,&ADD_S,&RXData);
                if(ADD_M.val32 == Add_Master)
                {
                    if(RXData.val32 == TICKET_RESET)
                    {
                        Led_Link_LostOff();
                        APP_UpdateState(APP_WAIT_FOR_TICKET);
                    }
                }
            }
            break;
        }


        case APP_ERROR:
        {
            //vide le FIFO
//            RXSize = GetReadSize(&descrFifoRX);
//            {
//                while (RXSize > 0)
//                {
//                    GetCharFromFifo(&descrFifoRX, &trash);
//                    RXSize --;
//                }
                APP_UpdateState(APP_WAIT_FOR_LINK);
//            }
            break;
        }

        case APP_RESET:
        {
            //Vide le FIFO
            RXSize = GetReadSize(&descrFifoRX);
            {
                while (RXSize > 0)
```

```c
                    {
                        GetCharFromFifo(&descrFifoRX, &trash);
                        RXSize --;
                    }
                    APP_UpdateState(APP_WAIT_FOR_TICKET);
                }
                break;
            }

            /* TODO: implement your application state machine.*/


            /* The default state should never be executed. */
            default:
            {
                /* TODO: Handle error in application's state machine. */
                break;
            }
        }
    }


    void __ISR(_CHANGE_NOTICE_VECTOR, ipl3AUTO) _IntHandlerChangeNotification(void)
    {

        //Bouton DECLINE
        if (PLIB_PORTS_PinGet (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_7))
        {
            flagTickPressed = true;
        }
        if (flagTickPressed)
        {
            if(!PLIB_PORTS_PinGet (PORTS_ID_0, PORT_CHANNEL_B, PORTS_BIT_POS_7))
            {
                appButtons.Btn_Tickets = true;
                flagTickPressed = false;
            }
        }

        PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_CHANGE_NOTICE_B);
    }

    void Blink_LED_ACC (void)
    {
        int I;
        ALL_LED_OFF;
        for(I = 0; I < 5000; I ++)
        {

        }
        ALL_LED_ON;
        for(I = 0; I < 5000; I ++)
        {

        }
        ALL_LED_OFF;
    }

    void ALL_LED_ON ()
    {
        Led_ReadyOn();
        Led_SendedOn();
        Led_Link_LostOn();
    }
    void ALL_LED_OFF ()
    {
        Led_ReadyOff();
        Led_SendedOff();
        Led_Link_LostOff();
    }

    /*******************************************************************************
     End of File
    */

```