```
/*******************************************************************************
PLAYER.H
 ******************************************************************************/
#ifndef Player_H
#define Player_H
#include "ES_Types.h"
bool Init_Player_XbeePIC( uint8_t Priority );
bool Post_Player_XbeePIC( ES_Event ThisEvent );
ES_Event Run_Player_XbeePIC( ES_Event ThisEvent );
#endif


/*---------------------------- PLAYER.C ----------------------------*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include <htc.h>
#include "SPI_Master.h"
#include "Player.h"
#include "pic16f690.h"
#include "bitdefs.h"
#include "PubFuncDefine.h"
#include "SCI_Receive_XbeePIC.h"
#include "SCI_Transmit_XbeePIC.h"
/*---------------------------- Module Defines ----------------------------*/
typedef enum
{
   WaitToLink, Paired, TagDetected
} PlayerState_t;
/*---------------------------- Module Variables ----------------------------*/
static PlayerState_t PlayerState;
static uint8_t PlayerPriority;
static unsigned char PairData = 0;
static unsigned char TAGINFO;
static unsigned char POS_O;
static unsigned char POS_D;
/*---------------------------- Module Code ----------------------------*/
bool Init_Player_XbeePIC( uint8_t Priority )
{
   ES_Event ThisEvent;
   PlayerPriority = Priority;
   PlayerState = WaitToLink;
   ThisEvent.EventType = ES_INIT;
   if (ES_PostToService( PlayerPriority, ThisEvent) == true)
      {
         return true;
      }
   else
      {
         return false;
      }
}
/*******************************************************************************/
```

```
bool Post_Player_XbeePIC( ES_Event ThisEvent )
{
   return ES_PostToService( PlayerPriority, ThisEvent);
}
/*****************************************************************************/
ES_Event Run_Player_XbeePIC( ES_Event ThisEvent )
{
   ES_Event PostEvent;
   ES_Event ReturnEvent;
   ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
   switch ( PlayerState )
      {
      case WaitToLink :
         if (ThisEvent.EventType == REQ_PAIR)
            {
               PairData =  ReceiveMessageFromXbee[6];
               if (((PairData | BIT0LO) == BIT0LO) && ((PairData | BIT1LO) ==
BIT1LO)&&((PairData & BIT2HI) == BIT2HI)&&((PairData & BIT3HI) == BIT3HI))
                  {
                     if ((((PairData & BIT4HI) == BIT4HI)&& (RC3 == 0))||(((PairData | BIT4LO)
== BIT4LO)&& (RC3 == 1)))
                        {
                           PAIR_ADDH = ReceiveMessageFromXbee[1];
                           PAIR_ADDL = ReceiveMessageFromXbee[2];
                           LiFKIMPICMessage[0] = 0x0F; // Request Energy level status
                           LiFKIMPICMessage[1] = 0x03; // SET LEVEL
                           LiFKIMPICMessage[2] = 0xFF; // TAGSIDE G/R
                           LiFKIMPICMessage[3] = 0xFF; // TAG OUT
                           LiFKIMPICMessage[4] = 0xFF; // TAG DETECTED
                           ServoPICMessage[0] = 0x01; //CTRL HDR
                           ES_Timer_InitTimer(HEARTBEAT_TIMER,3000);
                           ES_Timer_InitTimer(STATUS_TIMER,150);
                           STATUSM=0xFF;
                           PostEvent.EventType = PAIR_ACCEPTED;
                           Post_Transmit_XbeePIC(PostEvent);
                           Post_Master(PostEvent);
                           PlayerState = Paired;
                        }
                     else
                        {
                           FAIL_PAIR_ADDH = ReceiveMessageFromXbee[1];
                           FAIL_PAIR_ADDL = ReceiveMessageFromXbee[2];
                           PlayerState = WaitToLink;
                           PostEvent.EventType = MISMATCH_COLOR;
                           Post_Transmit_XbeePIC(PostEvent);
                        }
                  }
               else
                  {
                     PlayerState = WaitToLink;
                  }
            }
         else
```

```
            {
                PlayerState = WaitToLink;
            }
          break;
       case Paired :
          switch ( ThisEvent.EventType )
            {
            case RESET :   //RESET
               if (ReceiveMessageFromXbee[1] == PAIR_ADDH &&
ReceiveMessageFromXbee[2]==PAIR_ADDL)
                  {
                     ServoPICMessage[0] = 0x01; //CTRL
                     ServoPICMessage[1] = 0x00; //spd
                     ServoPICMessage[2] = 0x00; //dir
                     ServoPICMessage[3] = 0x00; //brk & kick
                     ServoPICMessage[4] = 0x00; //whim
                     ServoPICMessage[5] = 0x00; //tr
                     ServoPICMessage[6] = 0x00; //energy level
                     LiFKIMPICMessage[0] = 0x00; // Request Energy level status
                     LiFKIMPICMessage[1] = 0x00; // SET LEVEL
                     LiFKIMPICMessage[2] = 0xFF; // TAGSIDE G/R
                     LiFKIMPICMessage[3] = 0xFF; // tagout
                     LiFKIMPICMessage[4] = 0xFF; // TAG DETECTED
                     ES_Timer_InitTimer(RESET_TIMER,100); //FOR SPI RESET
                     PlayerState = WaitToLink;
                  }
               break;
            case TAG_OUT :   //control tagout
               if (ReceiveMessageFromXbee[1] == PAIR_ADDH &&
ReceiveMessageFromXbee[2]==PAIR_ADDL)
                  {
                     TAGINFO = ReceiveMessageFromXbee[6];
                     if (((TAGINFO | BIT0LO) == BIT0LO))
                        {
                           LiFKIMPICMessage[2] = 0x0D;
                        }
                     else if (((TAGINFO & BIT0HI) == BIT0HI))
                        {
                           LiFKIMPICMessage[2] = 0x0C;
                        }
                     if (((TAGINFO | BIT1LO) == BIT1LO)&&((TAGINFO | BIT2LO) == BIT2LO))
                        {
                           POS_O =0x08;
                           LiFKIMPICMessage[3] = POS_O;
                        }
                     else if (((TAGINFO | BIT1LO) == BIT1LO)&&((TAGINFO & BIT2HI) ==
BIT2HI))
                        {
                           POS_O =0x09;
                           LiFKIMPICMessage[3] = POS_O;
                        }
                     else if (((TAGINFO & BIT1HI) == BIT1HI)&&((TAGINFO | BIT2LO) ==
BIT2LO))
```

```
                    {
                        POS_O =0x0A;
                        LiFKIMPICMessage[3] = POS_O;
                    }
                else if (((TAGINFO & BIT1HI) == BIT1HI)&&((TAGINFO & BIT2HI) ==
BIT2HI))
                    {
                        POS_O =0x0B;
                        LiFKIMPICMessage[3] = POS_O;
                    }
                PlayerState = Paired;
            }
        break;
    case CTRL:
        if ((ReceiveMessageFromXbee[1] == PAIR_ADDH) &&
(ReceiveMessageFromXbee[2]==PAIR_ADDL))
            {
                ServoPICMessage[0] = 0x01; //CTRL
                ServoPICMessage[1] = ReceiveMessageFromXbee[6]; //spd
                ServoPICMessage[2] = ReceiveMessageFromXbee[7]; //dir
                ServoPICMessage[3] = ReceiveMessageFromXbee[8]; //brk & kick
                ServoPICMessage[4] = ReceiveMessageFromXbee[9]; //whim
                ServoPICMessage[5] = ReceiveMessageFromXbee[10]; //tr
                if ((ServoPICMessage[1] < -10)|(ServoPICMessage[3] > 0x3f))
                    {
                        LiFKIMPICMessage[1] = 0x00;
                    }
                else
                    {
                        LiFKIMPICMessage[1] = 0x03;
                    }
                ES_Timer_InitTimer(HEARTBEAT_TIMER,3000);
                PlayerState = Paired;
            }
        break;
    case TAG_DETECTED :   //control tagout
        TAGINFO = ReceiveMessageFromXbee[6];
        if (((TAGINFO | BIT0LO) == BIT0LO))
            {
                LiFKIMPICMessage[2] = 0x0D;
            }
        else if (((TAGINFO & BIT0HI) == BIT0HI))
            {
                LiFKIMPICMessage[2] = 0x0C;
            }
        if (((TAGINFO | BIT1LO) == BIT1LO)&&((TAGINFO | BIT2LO) == BIT2LO))
            {
                POS_D = 0x04;
                LiFKIMPICMessage[4] = POS_D;
            }
        else if (((TAGINFO | BIT1LO) == BIT1LO)&&((TAGINFO & BIT2HI) == BIT2HI))
            {
                POS_D = 0x05;
```

```
                LiFKIMPICMessage[4] = POS_D;
            }
          else if (((TAGINFO & BIT1HI) == BIT1HI)&&((TAGINFO | BIT2LO) == BIT2LO))
            {
                POS_D = 0x06;
                LiFKIMPICMessage[4] = POS_D;
            }
          else if (((TAGINFO & BIT1HI) == BIT1HI)&&((TAGINFO & BIT2HI) == BIT2HI))
            {
                POS_D = 0x07;
                LiFKIMPICMessage[4] = POS_D;
            }
          ES_Timer_InitTimer(TAG_DETECTED_TIMER,2000);
          PlayerState = TagDetected;
          break;
        case REQ_PAIR:   //control tagout
          PairData =  ReceiveMessageFromXbee[6];
          if (((PairData | BIT0LO) == BIT0LO) && ((PairData | BIT1LO) ==
BIT1LO)&&((PairData & BIT2HI) == BIT2HI)&&((PairData & BIT3HI) == BIT3HI))
            {
              if ((((PairData & BIT4HI) == BIT4HI)&& (RC3 == 0))||(((PairData | BIT4LO)
== BIT4LO)&& (RC3 == 1)))
                {
                  FAIL_PAIR_ADDH = ReceiveMessageFromXbee[1];
                  FAIL_PAIR_ADDL = ReceiveMessageFromXbee[2];
                  PostEvent.EventType = ALREADY_PAIRED;
                  Post_Transmit_XbeePIC(PostEvent);
                  PlayerState = Paired;
                }
            }
          break;
        case ES_TIMEOUT:   //control tagout
          if (ThisEvent.EventParam == HEARTBEAT_TIMER)
            {
                ServoPICMessage[0] = 0x01; //CTRL
                ServoPICMessage[1] = 0x00; //spd
                ServoPICMessage[2] = 0x00; //dir
                ServoPICMessage[3] = 0x00; //brk & kick
                ServoPICMessage[4] = 0x00; //whim
                ServoPICMessage[5] = 0x00; //tr
                ServoPICMessage[6] = 0x00; //energy level
                LiFKIMPICMessage[0] = 0x00; // Request Energy level status
                LiFKIMPICMessage[1] = 0x00; // SET LEVEL
                LiFKIMPICMessage[2] = 0xFF; // TAG DETECTED
                LiFKIMPICMessage[3] = 0xFF; // tagout
                LiFKIMPICMessage[4] = 0xFF; // TAGSIDE G/R
                ES_Timer_InitTimer(RESET_TIMER,100); //FOR SPI RESET
                PlayerState = WaitToLink;
            }
          else if (ThisEvent.EventParam == STATUS_TIMER)
            {
                PostEvent.EventType = TRANSMI_STATUS_TO_XBEE;
                //PostEvent.EventParam = 0;
```

```
                    Post_Transmit_XbeePIC(PostEvent);
                    ES_Timer_InitTimer(STATUS_TIMER,150);
                    PlayerState = Paired;
                }
              break;
            }
          break;
        case TagDetected :
          switch ( ThisEvent.EventType )
            {
            case TAG_OUT :   //control tagout
              if (ReceiveMessageFromXbee[1] == PAIR_ADDH &&
ReceiveMessageFromXbee[2]==PAIR_ADDL)
                {
                  TAGINFO = ReceiveMessageFromXbee[6];
                  if (((TAGINFO | BIT0LO) == BIT0LO))
                    {
                      LiFKIMPICMessage[2] = 0x0D;
                    }
                  else if (((TAGINFO & BIT0HI) == BIT0HI))
                    {
                      LiFKIMPICMessage[2] = 0x0C;
                    }
                  if (((TAGINFO | BIT1LO) == BIT1LO)&&((TAGINFO | BIT2LO) == BIT2LO))
                    {
                      POS_O =0x08;
                      LiFKIMPICMessage[3] = POS_O;
                    }
                  else if (((TAGINFO | BIT1LO) == BIT1LO)&&((TAGINFO & BIT2HI) ==
BIT2HI))
                    {
                      POS_O =0x09;
                      LiFKIMPICMessage[3] = POS_O;
                    }
                  else if (((TAGINFO & BIT1HI) == BIT1HI)&&((TAGINFO | BIT2LO) ==
BIT2LO))
                    {
                      POS_O =0x0A;
                      LiFKIMPICMessage[3] = POS_O;
                    }
                  else if (((TAGINFO & BIT1HI) == BIT1HI)&&((TAGINFO & BIT2HI) ==
BIT2HI))
                    {
                      POS_O =0x0B;
                      LiFKIMPICMessage[3] = POS_O;
                    }
                  PlayerState = TagDetected;
                }
              break;
            case RESET :   //RESET
              if (ReceiveMessageFromXbee[1] == PAIR_ADDH &&
ReceiveMessageFromXbee[2]==PAIR_ADDL)
                {
```

```
                ServoPICMessage[0] = 0x01; //CTRL
                ServoPICMessage[1] = 0x00; //spd
                ServoPICMessage[2] = 0x00; //dir
                ServoPICMessage[3] = 0x00; //brk & kick
                ServoPICMessage[4] = 0x00; //whim
                ServoPICMessage[5] = 0x00; //tr
                ServoPICMessage[6] = 0x00; //energy level
                LiFKIMPICMessage[0] = 0x00; // Request Energy level status
                LiFKIMPICMessage[1] = 0x00; // SET LEVEL
                LiFKIMPICMessage[2] = 0xFF; // TAGSIDE G/R
                LiFKIMPICMessage[3] = 0xFF; // tagout
                LiFKIMPICMessage[4] = 0xFF; // TAG DETECTED
                ES_Timer_InitTimer(RESET_TIMER,100);
                PlayerState = WaitToLink;
              }
            break;
          case CTRL:
            if (ReceiveMessageFromXbee[1] == PAIR_ADDH &&
ReceiveMessageFromXbee[2]==PAIR_ADDL)
              {
                ServoPICMessage[0] = 0x01;
                ServoPICMessage[1] = ReceiveMessageFromXbee[6];
                ServoPICMessage[2] = ReceiveMessageFromXbee[7];
                ServoPICMessage[3] = ReceiveMessageFromXbee[8];
                ServoPICMessage[4] = ReceiveMessageFromXbee[9];
                ServoPICMessage[5] = ReceiveMessageFromXbee[10];
                PlayerState = TagDetected;
                ES_Timer_InitTimer(HEARTBEAT_TIMER,3000);
              }
            break;
          case REQ_PAIR:   //control tagout
            PairData =  ReceiveMessageFromXbee[6];
            if (((PairData | BIT0LO) == BIT0LO) && ((PairData | BIT1LO) ==
BIT1LO)&&((PairData & BIT2HI) == BIT2HI)&&((PairData & BIT3HI) == BIT3HI))
              {
                if ((((PairData & BIT4HI) == BIT4HI)&& (RC3 == 0))||(((PairData | BIT4LO)
== BIT4LO)&& (RC3 == 1)))
                  {
                    FAIL_PAIR_ADDH = ReceiveMessageFromXbee[1];
                    FAIL_PAIR_ADDL = ReceiveMessageFromXbee[2];
                    PostEvent.EventType = ALREADY_PAIRED;
                    Post_Transmit_XbeePIC(PostEvent);
                    PlayerState = TagDetected;
                  }
              }
            break;
          case ES_TIMEOUT:   //control tagout
            if (ThisEvent.EventParam == TAG_DETECTED_TIMER)
              {
                PlayerState = Paired;
              }
            else if (ThisEvent.EventParam == HEARTBEAT_TIMER)
              {
```

```
                    ServoPICMessage[0] = 0x01; //CTRL
                    ServoPICMessage[1] = 0x00; //spd
                    ServoPICMessage[2] = 0x00; //dir
                    ServoPICMessage[3] = 0x00; //brk & kick
                    ServoPICMessage[4] = 0x00; //whim
                    ServoPICMessage[5] = 0x00; //tr
                    ServoPICMessage[6] = 0x00; //energy level
                    LiFKIMPICMessage[0] = 0x00; // Request Energy level status
                    LiFKIMPICMessage[1] = 0x00; // SET LEVEL
                    LiFKIMPICMessage[2] = 0xFF; // TAG DETECTED
                    LiFKIMPICMessage[3] = 0xFF; // tagout
                    LiFKIMPICMessage[4] = 0xFF; // TAGSIDE G/R
                    ES_Timer_InitTimer(RESET_TIMER,100);
                    PlayerState = WaitToLink;
                  }
                else if (ThisEvent.EventParam == STATUS_TIMER)
                  {
                    ES_Timer_InitTimer(STATUS_TIMER,150);
                    PostEvent.EventType = TRANSMI_STATUS_TO_XBEE;
                    Post_Transmit_XbeePIC(PostEvent);
                    PlayerState = TagDetected;
                  }
                break;
              }
            break;
        }
    return ReturnEvent;
}
```

```
/****************************************************************************
   EventCheckers.h
****************************************************************************/
#ifndef EventCheckers_H
#define EventCheckers_H
bool Check_XbeeDataReceived(void);
bool Check_XbeeDataTramsmit(void);
bool Check_SPISend(void);
#endif


/*---------------------------- EventCheckers.C ----------------------------*/
// this will pull in the symbolic definitions for events, which we will want
// to post in response to detecting events
#include "ES_Configure.h"
// this will get us the structure definition for events, which we will need
// in order to post events in response to detecting events
#include "ES_Events.h"
// if you want to use distribution lists then you need those function
// definitions too.
#include "ES_PostList.h"
// This include will pull in all of the headers from the service modules
```

```
// providing the prototypes for all of the post functions
#include "ES_ServiceHeaders.h"
// this test harness for the framework references the serial routines that
// are defined in ES_Port.c
#include "ES_Port.h"
// include our own prototypes to insure consistency between header &
// actual functionsdefinition
#include "EventCheckers.h"
//#include "PubFuncDefine.h"
#include "SCI_Transmit_XbeePIC.h"
/*****************************************************************************/
#define f 1
//#define LIFKIM_PIC_RequestBit RC0
/*****************************************************************************/
bool Check_XbeeDataReceived(void)
{
   ES_Event ThisEvent;
   //IF DATA RECEIVED
   if (RCIF ==1 )
      {
         ThisEvent.EventType = DATA_RECEIVED_FROM_XBEE;
         ThisEvent.EventParam = 0;
         Post_Receive_XbeePIC( ThisEvent );
         return true;
      }
   return false;
}
bool Check_XbeeDataTramsmit(void)
{
   ES_Event ThisEvent;
   //IF DATA TRANSMITTED
   if ((TXIF == 1) && (Query_Transmit_XbeePIC ()== TransmitToXbee_XbeeInProgress))
      {
         ThisEvent.EventType = DATA_TRANSMIT_TO_XBEE;
         Post_Transmit_XbeePIC( ThisEvent );
         return true;
      }
   return false;
}
bool Check_SPISend(void)
{
   ES_Event ThisEvent;
   //IF DATA RECEIVED FROM SPI
   if (SSPIF== 1)
      {
         if ((Query_Master()==Master_SendLiFKIM) || (Query_Master ()==Master_SendServo))
            {
               ThisEvent.EventType = SPI_READY_SEND;
               Post_Master( ThisEvent );
               return true;
            }
      }
   return false;
```

```
}
```

```
/*******************************************************************************
 SCIRECEIVE.H
 ******************************************************************************/
#ifndef SCI_Receive_XbeePIC_H
#define SCI_Receive_XbeePIC_H
#include "ES_Types.h"
bool Init_Receive_XbeePIC( uint8_t Priority );
bool Post_Receive_XbeePIC( ES_Event ThisEvent );
ES_Event Run_Receive_XbeePIC( ES_Event ThisEvent );
#endif


/*---------------------------- SCIRECEIVE.C ----------------------------*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include <htc.h>
#include "SPI_Master.h"
#include "player.h"
#include "pic16f690.h"
#include "bitdefs.h"
#include "PubFuncDefine.h"
#include "SCI_Receive_XbeePIC.h"
#include "SCI_Transmit_XbeePIC.h"
/*---------------------------- Module Defines ----------------------------*/
typedef enum
{
   ReceiveFromXbee_Idle = 0,
   ReceiveFromXbee_LengthHigh,
   ReceiveFromXbee_LengthLow,
   ReceiveFromXbee_Data,
   ReceiveFromXbee_Checksum,
} ReceiveFromXbeeState_t;
/*---------------------------- Module Variables ---------------------------*/
static ReceiveFromXbeeState_t ReceiveFromXbeeState;
static uint8_t XbeePICPriority;
static unsigned char ReceiveBrokenTime = 1500 ;
/*---------------------------- Module Code ----------------------------*/
bool Init_Receive_XbeePIC( uint8_t Priority )
{
   ES_Event ThisEvent;
   XbeePICPriority = Priority;
   ReceiveFromXbeeState = ReceiveFromXbee_Idle;
   ThisEvent.EventType = ES_INIT;
   ANSELH = 0x00;
   ANSEL = 0x00;
   TRISC3=1;                              //SWITCH
   TRISC4=0;          //RC4 output
   RC4=1;             //RC4 high
   TRISB5 = 1;        //Rx input
   TRISB7 = 0;                            //Tx output
```

```
    SPBRG = 129;                //D'129 SET BAUD RATE=9600
    SPBRGH = 0x00;
    TXSTA = 0x24 ;
    RCSTA = 0x90;               //Initialize RCSTA register
    if (ES_PostToService( XbeePICPriority, ThisEvent) == true)
       {
          return true;
       }
    else
       {
          return false;
       }
}
/*****************************************************************************/
bool Post_Receive_XbeePIC( ES_Event ThisEvent )
{
    return ES_PostToService( XbeePICPriority, ThisEvent);
}
/*****************************************************************************/
ES_Event Run_Receive_XbeePIC( ES_Event ThisEvent )
{
    ES_Event PostEvent;
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
    switch (ReceiveFromXbeeState)
       {
          //State: Waiting for new message
       case ReceiveFromXbee_Idle:
          if ( ThisEvent.EventType == DATA_RECEIVED_FROM_XBEE )
             {
                unsigned char ReceiveNewByteFromXbee = RCREG;
                if (ReceiveNewByteFromXbee == 0x7E)                                  //If
New Byte is 0x7E Start_Xbee
                   {
                      ReceiveFromXbeeState = ReceiveFromXbee_LengthHigh; //wait for
LengthHigh byte
                      ES_Timer_InitTimer(SCI_RECEIVE_TIMER,ReceiveBrokenTime);
                   }
             }
          break;
          //State: Waiting for LengthHigh byte
       case ReceiveFromXbee_LengthHigh:
          if ( ThisEvent.EventType == DATA_RECEIVED_FROM_XBEE )
             {
                unsigned char ReceiveNewByteFromXbee = RCREG;
                if (ReceiveNewByteFromXbee == 0x00)                                  //If New byte i
                   {
                      ReceiveFromXbeeState = ReceiveFromXbee_LengthLow; //wait for
LengthLow byte
                      ES_Timer_InitTimer(SCI_RECEIVE_TIMER,ReceiveBrokenTime);
                   }
                else
                   {
```

```
                    ReceiveFromXbeeState = ReceiveFromXbee_Idle; //Otherwise reset state
machine
                }
            }
        else if ( ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam ==
SCI_RECEIVE_TIMER )
            {
                ReceiveFromXbeeState = ReceiveFromXbee_Idle;
            }
        break;
        //State: Waiting for LengthLow byte
      case ReceiveFromXbee_LengthLow:
        if ( ThisEvent.EventType == DATA_RECEIVED_FROM_XBEE )
            {
                unsigned char ReceiveNewByteFromXbee = RCREG;
                ReceiveMessageFromXbeeLength = ReceiveNewByteFromXbee;      //On new byte,
store as message length
                ReceiveMessageFromXbeeChecksum = 0; //reset to prepare for new message,
reset checkusm
                ReceiveMessageFromXbeeIndex = 0;    //reset index
                ReceiveFromXbeeState = ReceiveFromXbee_Data; //Wait for data:
                ES_Timer_InitTimer(SCI_RECEIVE_TIMER,ReceiveBrokenTime);
            }
        else if ( ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam ==
SCI_RECEIVE_TIMER )
            {
                ReceiveFromXbeeState = ReceiveFromXbee_Idle;
            }
        break;
        //State: Waiting for Data bytes
      case ReceiveFromXbee_Data:
        if ( ThisEvent.EventType == DATA_RECEIVED_FROM_XBEE )
            {
                unsigned char ReceiveNewByteFromXbee = RCREG;
                ReceiveMessageFromXbee[ReceiveMessageFromXbeeIndex] =
ReceiveNewByteFromXbee;     //On new byte, store data
                ReceiveMessageFromXbeeChecksum += ReceiveNewByteFromXbee; //update
checksum:
                ReceiveMessageFromXbeeIndex++;  //increment index
                ES_Timer_InitTimer(SCI_RECEIVE_TIMER,ReceiveBrokenTime);
                //If received entire message
                if (ReceiveMessageFromXbeeIndex >= ReceiveMessageFromXbeeLength)
                  {
                    //wait for checksum
                    ReceiveFromXbeeState = ReceiveFromXbee_Checksum;
                    ES_Timer_InitTimer(SCI_RECEIVE_TIMER,ReceiveBrokenTime);
                  }
            }
        else if ( ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam ==
SCI_RECEIVE_TIMER )
            {
                ReceiveFromXbeeState = ReceiveFromXbee_Idle;
            }
```

```
            break;
            //State: Waiting for checksum
        case ReceiveFromXbee_Checksum:
            if ( ThisEvent.EventType == DATA_RECEIVED_FROM_XBEE )
                {
                    unsigned char ReceiveNewByteFromXbee = RCREG;
                    ReceiveMessageFromXbeeChecksum += ReceiveNewByteFromXbee; //On new
byte, update checksum
                    if (ReceiveMessageFromXbeeChecksum == 0xFF)          //Check if checksum is
valid
                      {
                        switch ( ReceiveMessageFromXbee[5] )
                          {
                          case 0x04 :   //RESET
                            PostEvent.EventType = RESET;
                            Post_Player_XbeePIC(PostEvent);
                            break;
                          case 0x02 :   //TAGOUT
                            PostEvent.EventType = TAG_OUT;
                            Post_Player_XbeePIC(PostEvent);
                            break;
                          case 0x01:    //CTRL
                            PostEvent.EventType = CTRL;
                            Post_Player_XbeePIC(PostEvent);
                            break;
                          case 0x07 :   //TAG_DETECTED
                            PostEvent.EventType = TAG_DETECTED;
                            Post_Player_XbeePIC(PostEvent);
                            break;
                          case 0x03 :   //REQ_PAIR
                            PostEvent.EventType = REQ_PAIR;
                            Post_Player_XbeePIC(PostEvent);
                            break;
                          }
                      }
                    ReceiveFromXbeeState = ReceiveFromXbee_Idle;
                }
            else if ( ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam ==
SCI_RECEIVE_TIMER )
                {
                    ReceiveFromXbeeState = ReceiveFromXbee_Idle;
                }
            break;
        }
    return ReturnEvent;
}
```

```
/****************************************************************************
  SCITRANSMIT.H
 ****************************************************************************/
#ifndef SCI_Transmit_XbeePIC_H
```

```h
#define SCI_Transmit_XbeePIC_H
#include "ES_Types.h"
typedef enum
{
    TransmitToXbee_Idle,
    TransmitToXbee_XbeeInProgress
} TransmitToXbeeState_t;
bool Init_Transmit_XbeePIC( uint8_t Priority );
bool Post_Transmit_XbeePIC( ES_Event ThisEvent );
ES_Event Run_Transmit_XbeePIC( ES_Event ThisEvent );
TransmitToXbeeState_t Query_Transmit_XbeePIC ( void );
#endif
```

```c
/*---------------------------- SCITRAMSMIT.C ----------------------------*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include <htc.h>
#include "SPI_Master.h"
#include "player.h"
#include "pic16f690.h"
#include "bitdefs.h"
#include "PubFuncDefine.h"
#include "SCI_Receive_XbeePIC.h"
#include "SCI_Transmit_XbeePIC.h"
/*---------------------------- Module Defines ----------------------------*/
/*---------------------------- Module Functions ----------------------------*/
static void StartXbeeTransmission(unsigned char Address_High, unsigned char Address_Low);
static void AddTransmissionData(unsigned char ByteToAdd);
static void FinishXbeeTransmission(void);
/*---------------------------- Module Variables ----------------------------*/
static TransmitToXbeeState_t TransmitToXbeeState ;
static uint8_t XbeePICPriority;
static unsigned char XbeeMessageForTransmit[11];
static unsigned char XbeeMessageForTransmitLength = 0;
static unsigned char XbeeMessageForTransmitIndex = 0;
static unsigned char XbeeMessageForTransmitChecksum;
/*---------------------------- Module Code ----------------------------*/
bool Init_Transmit_XbeePIC( uint8_t Priority )
{
    ES_Event ThisEvent;
    XbeePICPriority = Priority;
    TransmitToXbeeState = TransmitToXbee_Idle;
    ThisEvent.EventType = ES_INIT;
    ANSELH = 0x00;
    ANSEL = 0x00;
    TRISC3=1;                                   //SWITCH
    TRISC4=0;           //RC4 output
    RC4=1;              //RC4 high
    //RC3=0; ////////////////////////////////////////////
    TRISB5 = 1;         //Rx input
    TRISB7 = 0;                                 //Tx output
```

```
    SPBRG = 129;                //D'129 SET BAUD RATE=9600
    SPBRGH = 0x00;
    TXSTA = 0x24 ;
    RCSTA = 0x90;               //Initialize RCSTA register
    //INTCON = 0xC0;
    //PIE1 = 0x10 ;
    if (ES_PostToService( XbeePICPriority, ThisEvent) == true)
       {
          return true;
       }
    else
       {
          return false;
       }
}
/*****************************************************************************/
bool Post_Transmit_XbeePIC( ES_Event ThisEvent )
{
    return ES_PostToService( XbeePICPriority, ThisEvent);
}
/*****************************************************************************/
ES_Event Run_Transmit_XbeePIC( ES_Event ThisEvent )
{
    //ES_Event PostEvent;
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
    switch ( TransmitToXbeeState )
       {
       case TransmitToXbee_Idle :
          switch ( ThisEvent.EventType )
             {
             case PAIR_ACCEPTED :
                StartXbeeTransmission(PAIR_ADDH,PAIR_ADDL);
                AddTransmissionData(0x05);
                AddTransmissionData(0x01);
                FinishXbeeTransmission();
                XbeeMessageForTransmitIndex = 0;
                TransmitToXbeeState = TransmitToXbee_XbeeInProgress;
                break;
             case ALREADY_PAIRED :
                StartXbeeTransmission(FAIL_PAIR_ADDH,FAIL_PAIR_ADDL);
                AddTransmissionData(0x05);
                AddTransmissionData(0x04);
                FinishXbeeTransmission();
                XbeeMessageForTransmitIndex = 0;
                TransmitToXbeeState = TransmitToXbee_XbeeInProgress;
                break;
             case MISMATCH_COLOR :
                StartXbeeTransmission(FAIL_PAIR_ADDH,FAIL_PAIR_ADDL);
                AddTransmissionData(0x05);
                AddTransmissionData(0x02);
                FinishXbeeTransmission();
                XbeeMessageForTransmitIndex = 0;
```

```
                TransmitToXbeeState = TransmitToXbee_XbeeInProgress;
                break;
            case TRANSMI_STATUS_TO_XBEE :  //Send Status
                StartXbeeTransmission(PAIR_ADDH,PAIR_ADDL);
                AddTransmissionData(0x06);
                AddTransmissionData(STATUSM);
                FinishXbeeTransmission();
                XbeeMessageForTransmitIndex = 0;
                TransmitToXbeeState = TransmitToXbee_XbeeInProgress;
                break;
            }
        break;
    case TransmitToXbee_XbeeInProgress:
        if (ThisEvent.EventType == DATA_TRANSMIT_TO_XBEE)
            {
            if (XbeeMessageForTransmitIndex < XbeeMessageForTransmitLength)
                {
                TXREG = XbeeMessageForTransmit[XbeeMessageForTransmitIndex];
                XbeeMessageForTransmitIndex++;   //Send new byte, increment counter
                }
            else
                {
                XbeeMessageForTransmitIndex = 0;   //If we have finished sending
                TransmitToXbeeState = TransmitToXbee_Idle; //Return to Idle state
                }
            }
        break;
    }
  return ReturnEvent;
}
TransmitToXbeeState_t Query_Transmit_XbeePIC ( void )
{
  return(TransmitToXbeeState);
}
/*****************************************************************************
 private functions
 *****************************************************************************/
static void StartXbeeTransmission(unsigned char Address_High, unsigned char Address_Low)
{
  XbeeMessageForTransmitChecksum = 0xFF;//Checksum_Total; //Initial checksum
  XbeeMessageForTransmit[0] = 0x7E; //Start_Xbee;
  XbeeMessageForTransmit[1] = 0x00; //Length High
  XbeeMessageForTransmit[2] = 0x05; //Length Low, initial
  XbeeMessageForTransmit[3] = 0x01;//API_Identifier
  XbeeMessageForTransmitChecksum -= 0x01;
  XbeeMessageForTransmit[4] = 0x00;//FrameID
  XbeeMessageForTransmitChecksum -= 0x00;
  XbeeMessageForTransmit[5] = Address_High;
  XbeeMessageForTransmitChecksum -= Address_High;
  XbeeMessageForTransmit[6] = Address_Low;
  XbeeMessageForTransmitChecksum -= Address_Low;
  XbeeMessageForTransmit[7] = 0x00;//Option_None;
  XbeeMessageForTransmitChecksum -= 0x00;
```

```
    XbeeMessageForTransmitIndex = 8; //Next byte of message
    return;
}
static void AddTransmissionData(unsigned char ByteToAdd)
{
    XbeeMessageForTransmit[XbeeMessageForTransmitIndex] = ByteToAdd;  //add to message
    XbeeMessageForTransmit[2]++;   //increase length byte
    XbeeMessageForTransmitIndex++; //increment index
    XbeeMessageForTransmitChecksum -= ByteToAdd; //update checksum
    return;
}
static void FinishXbeeTransmission(void)
{
    //Add checksum byte:
    XbeeMessageForTransmit[XbeeMessageForTransmitIndex] =
XbeeMessageForTransmitChecksum;
    XbeeMessageForTransmitIndex++;
    XbeeMessageForTransmitLength = XbeeMessageForTransmitIndex; //length of message
    return;
}
```

```
/*****************************************************************************
SPIMASTER.H
 *****************************************************************************/
#ifndef SPI_Master_H
#define SPI_Master_H
#include "ES_Types.h"
typedef enum
{
    Master_Idle,
    Master_SendServo,
    Master_SendLiFKIM
} MasterState_t;
bool Init_Master( uint8_t Priority );
bool Post_Master( ES_Event ThisEvent );
ES_Event Run_Master( ES_Event ThisEvent );
MasterState_t Query_Master ( void );
#endif

*---------------------------- SPIMASTER.C ----------------------------*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include <htc.h>
#include "SPI_Master.h"
#include "player.h"
#include "pic16f690.h"
#include "bitdefs.h"
#include "PubFuncDefine.h"
#include "SCI_Receive_XbeePIC.h"
#include "SCI_Transmit_XbeePIC.h"
//__CONFIG( CP_OFF & WDTE_OFF & PWRTE_ON & FOSC_HS & MCLRE_OFF );
```

```
/*---------------------------- Module Defines ----------------------------*/
/*---------------------------- Module Variables ----------------------------*/
static MasterState_t MasterState;
static uint8_t MasterPriority;
static unsigned char LiFKIMPICMessageIndex = 0;
static unsigned char LiFKIMPICMessageLength = 9;
static unsigned char ServoPICMessageIndex = 0;
static unsigned char ServoPICMessageLength = 7;
static unsigned char SDATA;
//static unsigned char SPITIME1 = 5;
//static unsigned char SPITIME2 = 50;
/*---------------------------- Module Code ----------------------------*/
bool Init_Master( uint8_t Priority )
{
   ES_Event ThisEvent;
   MasterPriority = Priority;
   MasterState = Master_Idle;
   ThisEvent.EventType = ES_INIT;
   ANSELH = 0x00;
   ANSEL = 0x00;
   //Timer2 Init for SPI
   PR2 = 125;  //10kHz SPI rate, with 1:4 prescale below
   TRISC7 = 0;                          //SDO CLEAR output
   TRISB6 = 0;                          //SCK CLEAR <- Master output
   TRISB4 = 1;                          //SDI SET input
   TRISC6 = 0;                          //Lift subsystem PIC 2 SS output
   TRISC5 = 0;                          //propulsion & steering subsystem PIC 1 SS output
   //INTCON = 0xC0;
   //PIE1 = 0x38 ;
   //Configure SPI Status register
   T2CON = 0x05;
   SSPSTAT = 0x00;              //  0xxxxxxx    Input data sampled at middle of ouput time
   //  x0xxxxxx    CKE - Data transmitted on rising edge of SCK
   //T2CON = 0x05;
8          //  x0000xxx    1:1 postscale
   //  xxxxx1xx    TMR2 on
   //  xxxxxx01    1:4 prescale
   //setup SPI Control register
   SSPCON = 0x33;               //  xx1xxxxx    Enables SPI
   //  xxx1xxxx    CKP - clock idles high
   //  xxxx0011    SPI master, clock = TMR2 output / 2
   RC6 = 1; //Deselect slave initially
   RC5 = 1; //Dese
   LiFKIMPICMessage[5] = 0xAA;
   LiFKIMPICMessage[6] = 0xAA;
   LiFKIMPICMessage[7] = 0xAA;
   LiFKIMPICMessage[8] = 0xAA;
   if (ES_PostToService( MasterPriority, ThisEvent) == true)
      {
         return true;
      }
   else
      {
```

```
        return false;
      }
}
/*------------------------------------------------------------------------------*/
bool Post_Master( ES_Event ThisEvent )
{
   return ES_PostToService( MasterPriority, ThisEvent);
}
/*------------------------------------------------------------------------------*/
ES_Event Run_Master( ES_Event ThisEvent )
{
   //ES_Event PostEvent;
   ES_Event ReturnEvent;
   ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
   switch ( MasterState )
      {
      case Master_Idle :
         if ( ThisEvent.EventType == PAIR_ACCEPTED )
            {
               RC6 = 0 ;
               RC5 = 1 ;
               SSPBUF = 0x0E; // GO ACTIVE
               MasterState  =    Master_SendLiFKIM ;
            }
         break;
      case  Master_SendLiFKIM :
         if ( ThisEvent.EventType == SPI_READY_SEND )
            {
               SSPIF = 0;
               SDATA = SSPBUF; //Read data from other PIC
               switch (SDATA)
                  {
                  case 0x01 :
                     ServoPICMessage[6] = 0x01; //energy level
                     STATUSM &=~(1<<2);
                     STATUSM &=~(1<<1);
                     STATUSM |=1<<0;
                     break;
                  case 0x02 :
                     ServoPICMessage[6] = 0x02; //energy level
                     STATUSM &=~(1<<2);
                     STATUSM |=1<<1;
                     STATUSM &=~(1<<0);
                     break;
                  case 0x03 :
                     ServoPICMessage[6] = 0x03; //energy level
                     STATUSM &=~(1<<2);
                     STATUSM |=1<<1;
                     STATUSM |=1<<0;
                     break;
                  case 0x04 :
                     ServoPICMessage[6] = 0x04; //energy level
                     STATUSM |=1<<2;
```

```
            STATUSM &=~(1<<1);
            STATUSM &=~(1<<0);
            break;
        case 0x05 :
            ServoPICMessage[6] = 0x05; //energy level
            STATUSM |=1<<2;
            STATUSM &=~(1<<1);
            STATUSM |=1<<0;
            break;
        case 0x06 :
            ServoPICMessage[6] = 0x06; //energy level
            STATUSM |=1<<2;
            STATUSM |=1<<1;
            STATUSM &=~(1<<0);
            break;
        case 0x07 :
            ServoPICMessage[6] = 0x07; //energy level
            STATUSM |=1<<2;
            STATUSM |=1<<1;
            STATUSM |=1<<0;
            break;
        case 0x08 :
            STATUSM &=~(1<<4);
            STATUSM &=~(1<<3);
            break;
        case 0x09 :
            STATUSM &=~(1<<4);
            STATUSM |=1<<3;
            break;
        case 0x0A :
            STATUSM |=1<<4;
            STATUSM &=~(1<<3);
            break;
        case 0x0B :
            STATUSM |=1<<4;
            STATUSM |=1<<3;
            break;
        case 0x0F :
            STATUSM &=~(1<<7);
            break;
        }
    if (LiFKIMPICMessageIndex < LiFKIMPICMessageLength)
        {
            SSPBUF = LiFKIMPICMessage[LiFKIMPICMessageIndex];
            LiFKIMPICMessageIndex++;
            MasterState  = Master_SendLiFKIM;
        }
    else
        {
            SSPBUF = 0xAA; //DUMMY DATA
            RC6 = 1; //deselect LifKIM
            RC5 = 0; //select Servo
            LiFKIMPICMessageIndex = 0;
```

```
                    MasterState  = Master_SendServo;
                }
            }
        else if ((ThisEvent.EventType == ES_TIMEOUT )&& (ThisEvent.EventParam
==RESET_TIMER))
            {
                MasterState = Master_Idle;
            }
        break;
      case  Master_SendServo :
        if ( ThisEvent.EventType == SPI_READY_SEND )
            {
                SSPIF = 0;
                if (ServoPICMessageIndex < ServoPICMessageLength)
                {
                    SSPBUF = ServoPICMessage[ServoPICMessageIndex];
                    ServoPICMessageIndex++;
                    MasterState  = Master_SendServo;
                }
                else
                {
                    SSPBUF = 0xAA; ////DUMMY DATA
                    RC6 = 0; //select LifKIM
                    RC5 = 1;
                    ServoPICMessageIndex = 0;
                    MasterState  = Master_SendLiFKIM;
                }
            }
        else if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam
==RESET_TIMER)
            {
                MasterState = Master_Idle;
            }
        break;
      }
   return ReturnEvent;
}
MasterState_t Query_Master ( void )
{
   return(MasterState);
}
```

```
/****************************************************************************
 PubFunctDefinE.H
 ****************************************************************************/
#ifndef PubFuncDefine_H
#define PubFuncDefine_H
unsigned char PAIR_ADDH;
unsigned char PAIR_ADDL;
unsigned char FAIL_PAIR_ADDH;
unsigned char FAIL_PAIR_ADDL;
```

```
signed char ServoPICMessage[7];
unsigned char LiFKIMPICMessage[9];
unsigned char STATUSM = 0xFF;
unsigned char ReceiveMessageFromXbee[12];
unsigned char ReceiveMessageFromXbeeLength = 0;
unsigned char ReceiveMessageFromXbeeIndex = 0;
unsigned char ReceiveMessageFromXbeeChecksum;
#endif
```