```
/*---------------------------- Include Files ----------------------------*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_Timers.h"
#include "ES_Queue.h"
#include "InterpretSM.h"
#include "ES_Port.h"
#include <stdio.h>
#include "CheckValidSM.h"
#include "TransmitXbeeSM.h"

/*---------------------------- Module Defines ----------------------------*/
#define QUEUE_SIZE 8
#define MAX_FRAME_LENGTH 15

/*---------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this machine.They should be functions
   relevant to the behavior of this state machine
*/
unsigned int ReturnSourceAdd(void);
void InitSPIcommunication(void);
void LookingForController(void);
void ControllerFound(void);

/*---------------------------- Module Variables ----------------------------*/
// everybody needs a state variable, you may need others as well.
// type of state variable should match that of enum in header file
static InterpretState_t CurrentState;
static unsigned int SourceAdd = 0;
static unsigned int LastSourceAdd = 0;
static unsigned char SSP_ByteCounter = 0;

// Flags
static unsigned char AreWeZombie = 0;
static unsigned char LeftTurnDegraded = 0;
static unsigned char RightTurnDegraded = 0;
static unsigned char SpeedEnhanced = 0;
static unsigned char LRSwapped = 0;
static unsigned char FBSwapped = 0;

// Data bytes
static char SpeedByte = 0;
static char DirnByte = 0;
static unsigned char FlagData = 0;

static unsigned char *FrameArray;

// everybody needs a local store for the service's priority too!
static uint8_t MyPriority;


/*---------------------------- Module Code ----------------------------*/
```

```
/*****************************************************************************
 Initialization Function: Initializes communcation pins and register
*****************************************************************************/
boolean InitInterpretSM ( uint8_t Priority )
{
  ES_Event ThisEvent;
  InitSPIcommunication();
  MyPriority = Priority;
  CurrentState = InitInterpretPState;
  ThisEvent.EventType = ES_INIT;
  if (ES_PostToService( MyPriority, ThisEvent) == True)
  {
      return True;
  }else
  {
      return False;
  }
}

/*****************************************************************************
 Function
     PostInterpretSM
*****************************************************************************/
boolean PostInterpretSM( ES_Event ThisEvent )
{
  return ES_PostToService( MyPriority, ThisEvent);

}

/*****************************************************************************
 Function
    RunInterpretSM
*****************************************************************************/
ES_Event RunInterpretSM( ES_Event CurrentEvent )
{
  ES_Event ReturnEvent;

  ReturnEvent.EventType = ES_NO_EVENT; // assume no problems will occur

  switch ( CurrentState )
  {
      case InitInterpretPState :       // If current state is initial Psedudo State
          if ( CurrentEvent.EventType == ES_INIT )// Only respond to EF_Init
          {
              CurrentState = SearchingController;
              LookingForController();
           }
           break;       // end switch on CurrentState

      case SearchingController :
          switch ( CurrentEvent.EventType )
          {
              case ES_FRAME_REC :
```

```
        {
              FrameArray = ReturnDataArray();
              if(FrameArray[0] == 0x81 && FrameArray[5] == 0x03 && FrameArray[6] == 0xED)
              {
                  SourceAdd = ((unsigned int)FrameArray[1] << 8) + FrameArray[2];
                  if(SourceAdd != LastSourceAdd)
                  {
                      ES_Event ThisEvent;
                      ThisEvent.EventType = ES_CONTROLLER_FOUND;
                      PostTransmitXbeeSM(ThisEvent);
                      ControllerFound();
                      CurrentState  = InterpretingMsg;
                      ES_Timer_InitTimer(_1_SEC_TIMER_IP, ONE_SEC );
                  }
              }
         }
         break;

        case ES_POST_DIR_BYTE :
        {
              if(SSPIF)
              {
                  ES_Event ThisEvent3;
                  ThisEvent3.EventType = ES_POST_FLAG_BYTE;
                  SSPIF = 0;
                  SSPBUF = DirnByte;
                  PostInterpretSM(ThisEvent3);
              }
              else
              {
                  ES_Event ThisEvent2;
                  ThisEvent2.EventType = ES_POST_DIR_BYTE;
                  PostInterpretSM(ThisEvent2);
              }
         }
         break;

      case ES_POST_FLAG_BYTE :
        {
              if(SSPIF)
              {
                  SSPIF = 0;
                  SSPBUF = FlagData;
              }
              else
              {
                  ES_Event ThisEvent3;
                  ThisEvent3.EventType = ES_POST_FLAG_BYTE;
                  PostInterpretSM(ThisEvent3);
              }
         }
         break;
```

```
            case ES_BALLOON_POPPED :
            {
                AreWeZombie = 0b00100000;
             }
             break;   // end switch on CurrentEvent

        }
        break;        // end switch on Current State

    case InterpretingMsg :
        switch ( CurrentEvent.EventType )
        {
            case ES_FRAME_REC :
            {
                  FrameArray = ReturnDataArray();
                   if(FrameArray[0] == 0x81)
                  {
                      unsigned int TempSourceAdd = ((unsigned int)FrameArray[1] << 8) +
                      FrameArray[2];
                      if((FrameArray[5] == 0x02)  && (TempSourceAdd == SourceAdd))
                      {
                          ES_Event ThisEvent;
                          ES_Event ThisEvent2;
                          SpeedByte = (signed char)(FrameArray[6]);
                          DirnByte = (signed char)(FrameArray[7]);

                          if(!AreWeZombie) FlagData = 0;
                          else {FlagData = (AreWeZombie | LeftTurnDegraded | RightTurnDegraded
                           | SpeedEnhanced | LRSwapped | FBSwapped);}
                          if(FrameArray[8] & 0x01) RC1 = 1;
                          else{ RC1 = 0;}

                          SSPBUF = SpeedByte;
                          ThisEvent2.EventType = ES_POST_DIR_BYTE;
                          PostInterpretSM(ThisEvent2);

                          ThisEvent.EventType = ES_COMMAND_REC;
                          PostTransmitXbeeSM(ThisEvent);

                          ES_Timer_InitTimer(_1_SEC_TIMER_IP, ONE_SEC );
                      }
                      else if(FrameArray[5] == 0x01)
                      {
                          if(FrameArray[6] != 0) SpeedEnhanced = 0b00000001;
                          else { SpeedEnhanced = 0;}

                          if(FrameArray[7] != 0) LeftTurnDegraded = 0b00000010;
                          else { LeftTurnDegraded = 0;}

                          if(FrameArray[8] != 0) RightTurnDegraded = 0b00000100;
                          else { RightTurnDegraded = 0;}

                          if(FrameArray[9] & 0x0F) LRSwapped = 0b00001000;
```

```
                    else { LRSwapped = 0;}

                    if(FrameArray[9] & 0xF0) FBSwapped = 0b00010000;
                    else { FBSwapped = 0;}
                }
            }
        }
        break;

    case ES_POST_DIR_BYTE :
      {
            if(SSPIF)
            {
                ES_Event ThisEvent3;
                ThisEvent3.EventType = ES_POST_FLAG_BYTE;
                SSPIF = 0;
                SSPBUF = DirnByte;
                PostInterpretSM(ThisEvent3);
            }
            else
            {
                ES_Event ThisEvent2;
                ThisEvent2.EventType = ES_POST_DIR_BYTE;
                PostInterpretSM(ThisEvent2);
            }
       }
       break;

    case ES_POST_FLAG_BYTE :
      {
            if(SSPIF)
            {
                SSPIF = 0;
                SSPBUF = FlagData;
            }
            else
            {
                ES_Event ThisEvent3;
                ThisEvent3.EventType = ES_POST_FLAG_BYTE;
                PostInterpretSM(ThisEvent3);
            }
       }
       break;

    case ES_BALLOON_POPPED :
      {
            ES_Event ThisEvent2;

            AreWeZombie = 0b00100000;
            SpeedByte = 0;          //Set speed to 0
            DirnByte = 0;
            RC1 = 0;                //Set servo attack to 0
```

```
                    SSPBUF = SpeedByte;
                    ThisEvent2.EventType = ES_POST_DIR_BYTE;
                    PostInterpretSM(ThisEvent2);

                    LastSourceAdd = SourceAdd;          // If we become zombie, the same
                    controller should not be able to connnect
                    LookingForController();
                    CurrentState = SearchingController;
                }
                break;

            case ES_TIMEOUT :
              {
                    ES_Event ThisEvent2;

                    SpeedByte = 0;          //Set speed to 0
                    DirnByte = 0;
                    RC1 = 0;                //Set servo attack to 0

                    SSPBUF = SpeedByte;
                    ThisEvent2.EventType = ES_POST_DIR_BYTE;
                    PostInterpretSM(ThisEvent2);

                    LastSourceAdd = 0;      // If its a simple timeout ,the same controller may
                    connect
                    LookingForController();
                    CurrentState = SearchingController;
                }
                break;   // end switch on CurrentEvent
          }
          break;         // end switch on CurrentState
  }
  return ReturnEvent;
}


/****************************************************************************
 Function
     QueryInterpretSM
 Parameters
     None
 Returns
     InterpretState_t The current state of the state machine
 Description
     returns the current state of the state machine
 Notes

 Author
     J. Edward Carryer, 10/23/11, 22:27
****************************************************************************/
InterpretState_t QueryInterpretSM ( void )
{
   return(CurrentState);
```

```
}

/*****************************************************************************
 private functions
 *****************************************************************************/
unsigned int ReturnSourceAdd(void)
{
    return SourceAdd;
}

void InitSPIcommunication(void)
{

    CKP = 1;
    SSPM0 = 1;                          //(osc/16), CKP = 1, CKE = 0, i.e mode 3 , 2nd rising
    edge

    TRISC6 = 0;                         //output slave select line RC6 to digital, output, tied
    low
    RC6 = 0;

    TRISB4 = 1;                         //set RB4/SDI to digital input
    TRISC7 = 0;                         //set RC7/SDO to digital output
    TRISB6 = 0;                         //set RB6/SCK to digital output

    TRISC1 = 0;                         //Servo Pin set to output, initialized to 0
    RC1 = 0;

    TRISC2 = 0;                         //Set Beacon Signal pin to Output, initialized to 0
    RC2 = 0;

    SSPEN = 1;                          //Enable SPI communication
}

void LookingForController(void)
{
    RC2 = 1;
}
void ControllerFound(void)
{
    RC2 = 0;
}
```