```
/*******************************************************************************
 Module
   FAC_FSM.c

 Revision
   1.0.1

 Description
   This module maintains communication with the FAC throughout the duration of
   gameplay.

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 02/21/13       DYL     began editing for FAC_FSM

*******************************************************************************/
/*---------------------------- Include Files ----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine*/

#include <stdio.h>
#include <stdlib.h>
#include <mc9s12e128.h>
#include <S12e128bits.h>
#include <Bin_Const.h>
#include <termio.h>
#include <hidef.h>
#include "S12eVec.h"

#include "E128_PWM.h"         //has all prescale definitions
#include "E128_SPI.h"
#include "E128_Servo.h"
#include "FAC_FSM.h"
#include "NavigationFSM.h"
#include "AlignPPService.h"
#include "DriveTrainService.h"
#include "ArtilleryFSM.h"
#include "StrategyFSM.h"


/*---------------------------- Module Defines ----------------------------*/
#define SS_DIR          DDRS_DDRS7
#define SS_PORT         PTS_PTS7
#define UPDATE_TIME     25 //updating FAC: change as necessary



/*---------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this service.They should be functions
   relevant to the behavior of this service*/


/*---------------------------- Module Variables ----------------------------*/
// everybody needs a state variable, you may need others as well.
// type of state variable should match htat of enum in header file
static FACState_t CurrentState;
static unsigned char i = 3;
```

```
static unsigned char k = 0;

static unsigned char Message;
static unsigned char NullMessage = 0x00;
static unsigned char OutgoingMessage;
static unsigned char Array[40][3];
static unsigned char InPlayArray[40] =
{
   0,0,1,1,1,1,1,1,1,1,
   1,1,1,1,1,1,1,1,1,1,
   1,1,1,1,1,1,1,1,1,1,
   1,1,1,1,1,1,1,1,1,1
}; //table of true/false for items that exist on the field

static unsigned char SelfColor;
static unsigned char EnemyColor;

// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;



/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
    InitFAC_FSM

 Parameters
    uint8_t : the priorty of this service

 Returns
    boolean, False if error in initialization, True otherwise

 Description
    Saves away the priority, sets up the initial transition and does any
    other required initialization for this state machine
 Notes

 Author
          Debbie Li and Jina Wang, 2/27/2013
****************************************************************************/
boolean InitFAC_FSM ( uint8_t Priority )
{
   ES_Event ThisEvent;

   MyPriority = Priority;

   // put us into the WaitingForUpdate
   CurrentState = WaitingForUpdate;

   // start FAC 1000ms update timer
   ES_Timer_InitTimer(FACUPDATE_TIMER, UPDATE_TIME);

   // set SS_PORT as output
   SS_DIR = HI;

   // post the initial transition event
   ThisEvent.EventType = ES_INIT;
   if (ES_PostToService( MyPriority, ThisEvent) == True)
      {
```

```
        return True;
      }
    else
      {
        return False;
      }
}

/****************************************************************************
 Function
    PostFAC_FSM

 Parameters
    EF_Event ThisEvent , the event to post to the queue

 Returns
    boolean False if the Enqueue operation failed, True otherwise

 Description
    Posts an event to this state machine's queue
 Notes

 Author
    Debbie Li and Jina Wang, 2/27/2013
****************************************************************************/
boolean PostFAC_FSM( ES_Event ThisEvent )
{
   return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
   RunFAC_FSM

 Parameters
  ES_Event : the event to process

 Returns
  ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
  add your description here
 Notes
  uses nested switch/case to implement the machine.
 Author
  Debbie Li and Jina Wang, 2/27/2013
****************************************************************************/
ES_Event RunFAC_FSM( ES_Event ThisEvent )
{
   ES_Event ReturnEvent;
   ReturnEvent.EventType = ES_NO_EVENT; // assume no errors


   /****************************************************************************

                    BEGIN STATE MACHINE CODE

   ****************************************************************************/

   switch ( CurrentState )
```

```
    {

    case ( WaitingForUpdate ) :
      if ((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam ==
FACUPDATE_TIMER))
        {
          i=2; //i=3 start message at bot 3, for i=2 query game status
          ES_Timer_InitTimer(SS_TIMER, 2);  //2 mS slave select timer
          CurrentState = PausingbtXfer;
        }
      break; // break WaitingForUpdate

    case ( PausingbtXfer ) :
      if ((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam == SS_TIMER))
        {
          Message = i;
          k = 0; //reset array index
          SS_PORT = LO;
          if (i == 2)
            {
              OutgoingMessage = 0x3F; //query game status
            }
          else
            {
              OutgoingMessage = (Message | 0xC0); //query bot or ship location
            }
          WriteMessage(OutgoingMessage);
          CurrentState = XferingByte1;
        }
      break; // PausingbtXfer

    case ( XferingByte1 ) :
      if (ThisEvent.EventType == SPIF_SET)
        {
          //ReadMessage(); //should return 0x00
          WriteMessage(NullMessage); //write the next message
          CurrentState = XferingByte2;
        }
      break; //break XferingByte2

    case ( XferingByte2 ) :
      if (ThisEvent.EventType == SPIF_SET)
        {
          //ReadMessage(); //should return 0x0FF
          WriteMessage(NullMessage);
          CurrentState = XferingByte3;
        }
      break; //break XferingByte2

    case ( XferingByte3 ) :
      if (ThisEvent.EventType == SPIF_SET)
        {
          Array[i][k] = ThisEvent.EventParam; //save x-coordinate
          k++; //increment k for next index in array
          WriteMessage(NullMessage);
          CurrentState = XferingByte4;
        }
      break; //break XferingByte3

    case ( XferingByte4 ) :
```

```
        if (ThisEvent.EventType == SPIF_SET)
          {
            Array[i][k] = ThisEvent.EventParam; //save y-coordinate
            k++; //increment k for next index in array
            WriteMessage(NullMessage);
            CurrentState = XferingByte5;
          }
        break; //break XferingByte4

      case ( XferingByte5 ) :
        if (ThisEvent.EventType == SPIF_SET)
          {
            Array[i][k] = ThisEvent.EventParam; //save theta
            k++; //increment k for next index in array
            SS_PORT = HI; //deassert SlaveSelect, done with 5-byte xfer
            if ( i <= 39 )
              {
                if ( (Array[i][0] == 0x00) && (i > 2) && (i != QueryEnemy()) && (i != SelfNum))
                  {
                    //depending on which bot we're playing against.
                    InPlayArray[i] = 0; //if ship/bot does not exist, delete from InPlayArray
                  }

                //Determine Next Message
                while (( InPlayArray[i+1] == 0 ) && ( (i+1) <= 39 ) )
                  {
                    i++; //increment i for next message

                  }
                i++;

                //printf("There is data for i = %i\r\n", i);

                ES_Timer_InitTimer(SS_TIMER, 2); //wait 2ms before next 5-byte xfer
                CurrentState = PausingbtXfer;
              }


            if ( i > 39) //done updating all bots & ships
              {
                //update the FAC at set UPDATE_TIME interval
                ES_Timer_InitTimer(FACUPDATE_TIMER, UPDATE_TIME);
                //post to Navigation done updating all ships
                ThisEvent.EventType = FAC_UPDATED;
                ThisEvent.EventParam = 0;
                PostNavigationFSM(ThisEvent);
                if (QueryStrategyFSM() == GatheringIntel | QueryStrategyFSM() == GameOver)
                  {
                    PostStrategyFSM(ThisEvent);
                  }



                //printf("Current position: x,y,theta %i,%i,%i \r\n", QueryX(SelfNum), QueryY(SelfNum),
QueryTheta(SelfNum));
                //printf("SelfColor = %i because Self_Xloc = %i Self_Yloc = %i \r\n", QueryColor(),
QueryX(SelfNum), QueryY(SelfNum));
                CurrentState = WaitingForUpdate;
              }
          }
```

```
            break; //break XferingByte5

        }// End switch( CurrentState )
    return ReturnEvent;
}

/****************************************************************************

                    END STATE MACHINE CODE

****************************************************************************/

/*---------------------------- Public Functions ----------------------------*/

unsigned char QueryGameState(void)
{
    static unsigned char GameState;
    GameState = QueryTheta(2);

    return GameState;
}


void GameStartIntel(void)
{
    //determine own and enemy  position, determine number of enemy ships
    if (QueryX(SelfNum) < 127)
        {
            SelfColor = RED;        //light corresponding LED
            EnemyColor = BLUE;      //light corresponding LED
            //DetermineNumEnemyShips();
        }
    else
        {
            SelfColor = BLUE;
            EnemyColor = RED;
            //DetermineNumEnemyShips();
        }
}


unsigned char QueryX(unsigned char ShipNum)
{
    return Array[ShipNum][0]; //x-coord
}


unsigned char QueryY(unsigned char ShipNum)
{
    return Array[ShipNum][1]; //y-coord
}


unsigned char QueryTheta(unsigned char ShipNum)
{
    return Array[ShipNum][2]; //theta
}


unsigned char DetermineEnemy(void)
```

```
{
    unsigned char i;
    unsigned char EnemyNum;

    for (i = 3; i < 20; i++)
        {
            if ((QueryX(i) != 0x00) && (i != SelfNum))
                {
                    EnemyNum = i;
                    return EnemyNum;
                }
        }



}


unsigned char QueryColor(void)
{
    return SelfColor;
}


unsigned char DetermineShip2Target(unsigned char SelfColor_)
{

    //RED SHIPS [20-29]
    //BLUE SHIPS [30-39]

    unsigned char i;
    unsigned char TargetNum = 0;

    if (SelfColor_ == RED)
        {
            for (i = 30; i < 40; i++)
                {
                    if (QueryX(i) != 0x00)
                        {
                            TargetNum = i;
                            return TargetNum;
                        }
                }
        }

    else
        {
            for (i = 20; i < 30; i++)
                {
                    if (QueryX(i) != 0x00)
                        {
                            TargetNum = i;
                            return TargetNum;
                        }
                }

        }

    return TargetNum;
}
```