```c
/****************************************************************************
 Module
   DogMasterSM.c

 Revision
   1.0.1

 Description
   The receiving state machine for the Farmer

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 05/20/17 1:51    bag               created for the project
****************************************************************************/
/*----------------------------- 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 "ES_Configure.h"
#include "ES_Framework.h"
#include "DogMasterSM.h"
#include "DogTXSM.h"
#include "DogRXSM.h"
#include "Constants.h"
#include "Hardware.h"
#include "DiscoBallSM.h"

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_nvic.h"
#include "inc/hw_uart.h"
#include "inc/hw_sysctl.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"      // Define PART_TM4C123GH6PM in project
#include "driverlib/gpio.h"
#include "driverlib/uart.h"

#define HARD_CODE_DOG_TAG 2

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

static void HandleReq( void );
static void HandleCtrl( void );

/*---------------------------- 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 DogMasterState_t CurrentState;
static uint8_t MyPriority;
static uint8_t DogSelect;
static bool PeripheralActive;


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

```
 Function
     InitDogMasterSM

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, 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
     Matthew W Miller, 5/13/2017, 17:31
****************************************************************************/
bool InitDogMasterSM(uint8_t Priority)
{
      // state is unpaired
      CurrentState = Unpaired;
      // post entry event to self
      ES_Event EntryEvent;
      EntryEvent.EventType = ES_ENTRY;
      // set priority
      MyPriority = Priority;

      //make sure lift fan is disabled
      sendToPIC(LIFT_FAN_OFF);
      PeripheralActive = false;

      if (PostDogMasterSM(EntryEvent))
      {
            return true;
      }
      else
      {
            return false;
      }
}

/****************************************************************************
 Function
     PostDogMasterSM

 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
     J. Edward Carryer, 10/23/11, 19:25
****************************************************************************/
bool PostDogMasterSM(ES_Event ThisEvent)
{
      // post event
      return ES_PostToService(MyPriority, ThisEvent);
}
```

```
/****************************************************************************
 Function
    RunDogMasterSM

 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
   Matthew Miller, 05/13/17, 17:54
****************************************************************************/
ES_Event RunDogMasterSM(ES_Event ThisEvent)
{
      // set return event
      ES_Event ReturnEvent;
      ReturnEvent.EventType = ES_NO_EVENT;

      // next state is current state
      DogMasterState_t NextState;
      NextState = CurrentState;
      //printf("DogMasterCurrentState = %i\r\n",CurrentState);

      // switch through states
      switch(CurrentState)
      {
            // if current state is unpaired
            case Unpaired:
                  // if event is entry
            printf("Dog Master SM -- Unpaired State -- Top\r\n");
                  if(ThisEvent.EventType == ES_ENTRY)
                  {
                        // stop electromechanical indicator
                        // clear LED active
                        // call LED setter
                        // turn thrust fan off
                        // set all brakes inactive
                        // call brake setter
                        // turn lift fan off
                        //printf("Dog Master SM -- Unpaired State -- Entry
Event\r\n");
                  }

                  // else if the event is ES_MESSAGE_REC and the header is a PAIR_REQ
and the API is 81 and dog tag is correct
                  else if(ThisEvent.EventType == ES_MESSAGE_REC && getHeader() ==
REQ_2_PAIR && getHardwareDogTag() == getSoftwareDogTag())
                  {
                        // next state is Wait2Pair
                        NextState = Wait2Pair;
                        printf("Dog Master SM -- Unpaired State -- Broadcast
Received\r\n");
                        HandleReq();

                        //start 1s connection timer
                        ES_Timer_InitTimer(CONN_TIMER, CONNECTION_TIME);

                  }
```

```
                    break;

                // else if current state is Wait2Pair
                case Wait2Pair:
                //printf("Dog Master SM -- Wait2Pair State -- Top\r\n");
                    //if event is Lost connection
                    if((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam
== CONN_TIMER))
                    {
                        printf("Dog Master SM -- Wait2Pair State -- Connection
Lost\r\n");
                        // next state is Unpaired
                        NextState = Unpaired;

                        //Clear the data array
                        ClearDataArray();

                        // post entry event to self
                        ES_Event NewEvent;
                        NewEvent.EventType = ES_ENTRY;
                        PostDogMasterSM(NewEvent);

                        //Post a lost connection event to the receive service
                        NewEvent.EventType = ES_LOST_CONNECTION;
                        PostDogRXSM(NewEvent);

                    }

                    // else if event is pair successful
                    else if(ThisEvent.EventType == ES_MESSAGE_REC && getHeader() ==
ENCR_KEY && (getDestFarmerAddressLSB() == getLSBAddress() &&
getDestFarmerAddressMSB() == getMSBAddress()))
                    {
                        printf("Dog Master SM -- Wait2Pair State -- Got Encryption
Key\r\n");
                        //Store the Encryption Key
                        StoreEncr();

                        // set LED active
                        // Call LED setter
                        // turn on electromechanical indicator
                        ES_Event NewEvent;
                        NewEvent.EventType = ES_PAIR_SUCCESSFUL;
                        PostDiscoBallSM(NewEvent);

                        // start lift fan
                        sendToPIC(LIFT_FAN_ON);

                        // next state is Paired
                        NextState = Paired;

                        //Call setDogDataHeader with STATUS parameter
                        setDogDataHeader(STATUS);

                        //Post transmit STATUS Event to TX_SM

                        NewEvent.EventType = ES_SEND_RESPONSE;
                        PostDogTXSM(NewEvent);

                        //restart 1s connection timer
                        ES_Timer_InitTimer(CONN_TIMER, CONNECTION_TIME);
                    }
```

```
                  break;

            // else if state is paired
            case Paired:
            //printf("Dog Master SM -- Paired State -- Top\r\n");
                  //if event is Lost connection or timeout
                  if((ThisEvent.EventType == ES_TIMEOUT) && (ThisEvent.EventParam
== CONN_TIMER))
                  {
                        printf("Dog Master SM -- Paired State -- Connection
Lost\r\n");

                        // turn thrust fan off
                        SetThrustFan(127);

                        // clear LED active
                        // call LED setter

                        // set all brakes inactive
                        SetLeftBrakePosition(LEFT_SERVO_UP);
                        SetRightBrakePosition(RIGHT_SERVO_UP);

                        // call brake setter

                        // turn lift fan off
                        sendToPIC(LIFT_FAN_OFF);
                        PeripheralActive = false;

                        //Clear the data array
                        ClearDataArray();

                        // next state is Unpaired
                        NextState = Unpaired;

                        // post entry event to self
                        ES_Event NewEvent;
                        NewEvent.EventType = ES_ENTRY;
                        PostDogMasterSM(NewEvent);

                        //Let the receive service know we have lost connection
                        NewEvent.EventType = ES_LOST_CONNECTION;
                        PostDogRXSM(NewEvent);

                        // stop electromechanical indicator
                        PostDiscoBallSM(NewEvent);
                  }

                  //If event is ES_MESSAGE_REC and encryption is synchronized and
same address
                  else if(ThisEvent.EventType == ES_MESSAGE_REC &&
(getDestFarmerAddressLSB() == getLSBAddress() && getDestFarmerAddressMSB() ==
getMSBAddress()))
                  {
                        DecryptData();
                        if(getHeader() == CTRL)
                        {
                              HandleCtrl();
                        }
                        else
                        {
                              printf("Dog Master SM -- Paired State -- Encryption
Reset\r\n");

                              //Send an ENCR_RESET mess to TX to send to farmer
```

```
                                setDogDataHeader(ENCR_RESET);

                                //Reset Encryption
                                ResetEncr();

                                //Post transmit ENCR_RESET Event to TX_SM
                                ES_Event NewEvent;
                                NewEvent.EventType = ES_SEND_RESPONSE;
                                PostDogTXSM(NewEvent);

                          }
                          //restart 1s connection timer
                          ES_Timer_InitTimer(CONN_TIMER, CONNECTION_TIME);
                    }

      }
      CurrentState = NextState;
      return ReturnEvent;
}

static void HandleCtrl( void ){
      printf("Dog Master SM -- Handle Control -- Top\r\n");

      //TODO: Restart the 1 second timer

      //Call setDogDataHeader with STATUS parameter
      setDogDataHeader(STATUS);
      //Post transmit STATUS Event to TX_SM
      ES_Event NewEvent;
      NewEvent.EventType = ES_SEND_RESPONSE;
      PostDogTXSM(NewEvent);


      //set the thrust fan to the value that was sent over Xbee
      SetThrustFan(getMoveData());


      //if TurnData is greater than 127
      if(getBrakeData() > 0)
      {
            //put down both servos
            SetLeftBrakePosition(LEFT_SERVO_DOWN);
            SetRightBrakePosition(RIGHT_SERVO_DOWN);
      }
      else if(getTurnData() > LEFT_TURN_THRESHOLD)
      {
            // TODO: Turn left servo on
            printf("Turn left Servo\r\n");
            // move right servo up
            SetRightBrakePosition(RIGHT_SERVO_UP);
            // move left servo to brake position
            SetLeftBrakePosition(LEFT_SERVO_DOWN);

      }//elseif TurnData is less than 127
      else if(getTurnData() < RIGHT_TURN_THRESHOLD){
            // TODO: Turn right servo on
            printf("Turn right Servo\r\n");
            // move left servo up
            SetLeftBrakePosition(LEFT_SERVO_UP);
            // move right servo to brake position
            SetRightBrakePosition(RIGHT_SERVO_DOWN);
      }
      else //we don't want to turn, so move both servos up
      {
```

```c
            SetLeftBrakePosition(LEFT_SERVO_UP);
            SetRightBrakePosition(RIGHT_SERVO_UP);
      }

      //if PerData is greater than 0
      if(getPerData() > 0)
      {
            // TODO: Toggle peripheral functionality (lift fan maybe)
            if(!PeripheralActive)
            {
                  PeripheralActive = true;
                  sendToPIC(LIFT_FAN_OFF);
                  printf("Peripheral functionality Toggled ON\r\n");
            }
            else
            {
                  PeripheralActive = false;
                  sendToPIC(LIFT_FAN_ON);
                  printf("Peripheral functionality Toggled OFF\r\n");
            }
      }


      ClearDataArray();
}

uint8_t getHardwareDogTag( void ){
      //TODO: Determine which dog we are maybe using ADMulti
      //return DogSelect;
      return ReadDOGTag();
}

static void HandleReq( void ){
      printf("Dog RX SM -- Handle Request -- Top\r\n");
      //TODO: START ONE SECOND TIMER\

      //Set Destination address of Farmer
      setDestFarmerAddress(getMSBAddress(),getLSBAddress());

      //Call setDogDataHeader with PAIR_ACK parameter
      setDogDataHeader(PAIR_ACK);

      //Post transmit PAIR_ACK Event to TX_SM
      ES_Event NewEvent;
      NewEvent.EventType = ES_SEND_RESPONSE;
      PostDogTXSM(NewEvent);
}
```