```c
/*******************************************************************************
 Module
   Master_SM.c

 Description
   This controls the master state machine for PAC.
*******************************************************************************/
/*----------------------------- 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 "Master_SM.h"
#include "Helpers.h"
#include "Definitions.h"
#include "UART_Service.h"
#include "Transmit_Service.h"
#include "Receive_SM.h"
#include "Outputs_Service.h"
/*----------------------------- Module Defines ----------------------------*/



/*---------------------------- 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 MasterState_t CurrentState;
static uint8_t status = 0x00;
static uint16_t PartnerAddr = NULL;
static bool decryption_failure;
static bool is_paired;
static uint8_t MyPriority;


/*------------------------------ Module Code ------------------------------*/
/*******************************************************************************
 Function
     Initialize the Master State Machine

 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
*******************************/
bool InitMaster_SM ( uint8_t Priority )
{
    ES_Event ThisEvent;

    MyPriority = Priority;

    // put us into the Initial PseudoState
    CurrentState = _Init;
    //initialize bools for pair status and decryption failure
    decryption_failure = false;
    is_paired = false;
```

```
    // post the initial transition event
    ThisEvent.EventType = ES_INIT;

    if (ES_PostToService( MyPriority, ThisEvent) == true)
    {
        return true;
    }
    else
    {
        return false;
    }
}

/****************************************************************************
 Function
     PostMaster
****************************************************************************/
bool PostMaster_SM( ES_Event ThisEvent )
{
    return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunMaster_SM

 Parameters
   ES_Event : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
   This Function runs through the state machine for the main controller
****************************************************************************/
ES_Event RunMaster_SM( ES_Event ThisEvent )
{
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    switch ( CurrentState )
    {
    case _Init :       // If current state is initial Psedudo State
        if ( ThisEvent.EventType == ES_INIT )// only respond to ES_Init
        {
            // now put the machine into the actual initial state
            CurrentState = _Unpaired;
            commandCurrentLift(LIFT_OFF);//make sure we start in static position
            disengageBrakes();
        }
        break;
    case _Unpaired :
        if ( ThisEvent.EventType == ES_PAIR_REQUEST )
        {
            if (ThisEvent.EventParam != PartnerAddr) //as long as this request
                is coming from new address, not last one...
            {
                ES_Timer_InitTimer(PAIRING_TIMER, PAIRING_TIME);//start 45 s
                                    pairing timer
                ES_Timer_InitTimer(TRANSMIT_TIMER, TRANSMIT_TIME);//start 1s
                                    transmit timeout timer
```

```
            commandCurrentLift(LIFT_ON);

            //Update DMC
            setCurrentColor(getColorBit());
            setCurrentPairingStatus(PAIRED_BIT);
            ES_Timer_InitTimer(DMC_TIMING_TIMER, DMC_TIMING_TIMER_TIME);
                               //start DMC Timer

            is_paired = true;//to transmit status with pairing bit set
            PartnerAddr = ThisEvent.EventParam;//store address of this PAC
                          in module variable
            uint8_t Destination_LSB = (PartnerAddr & 0xff); //set partner
                                      address in TxmitService
            uint8_t Destination_MSB = PartnerAddr >> 8;
            setTransmitPacketDestinationMSB(Destination_MSB);
            setTransmitPacketDestinationLSB(Destination_LSB);
            sendStatus(); //send status and change state
            CurrentState = _Waiting4Encryption;
        }
    }
    break;
case _Waiting4Encryption :
    if ( ThisEvent.EventType == ES_ENCRYPTION_RECVD )//if we get a good
         encryption
    {
        ES_Timer_InitTimer(TRANSMIT_TIMER, TRANSMIT_TIME);//restart 1s
                           transmit timeout timer
        sendStatus();//transmit status with pairing bit set
        CurrentState = _Waiting4Control;
        setEncryptionIndex();
    }
    if (ThisEvent.EventType == ES_PAIR_REQUEST) //if they send another
        request because they didn't get our last ack status
    {
        if (ThisEvent.EventParam == PartnerAddr) //as long as this request
            is coming from new address, not last one
        {
            ES_Timer_InitTimer(TRANSMIT_TIMER, TRANSMIT_TIME);//start 1s
                               transmit timeout timer
            sendStatus();
        }
    }
    if ( ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam ==
         TRANSMIT_TIMER)//if we get a transit timeout
    {
        commandCurrentLift(LIFT_OFF);//deactivate lift fan
        ES_Timer_StopTimer(PAIRING_TIMER);//disable all timers
        ES_Timer_StopTimer(TRANSMIT_TIMER);
        ES_Timer_StopTimer(DMC_TIMING_TIMER);
        setCurrentColor(COLOR_DEACTIVATE);//set parameters (color and pair
                        status)
        setCurrentPairingStatus(!PAIRED_BIT);
        is_paired = false;//transmit status with pairing bit cleared
        sendStatus();//send and go to unpaired state
        CurrentState = _Unpaired;
    }
    break;
case _Waiting4Control :
    if ( ThisEvent.EventType == ES_CTRL_THRUST )//individual events for
         thrust, orientation, and special action
    {
        if (!isSpecialAction()) //as long as we aren't in special action
```

```
    {
        commandCurrentThrust(ThisEvent.EventParam);
    }
}
if ( ThisEvent.EventType == ES_CTRL_ORIENT )//orientation (left/right)
{
    if (!isSpecialAction())
    {
        commandCurrentOrientation(ThisEvent.EventParam);
    }
}
if ( ThisEvent.EventType == ES_CTRL_SPECIAL )//8 bits for the special
     action
{
    //check and send brake
    if ((ThisEvent.EventParam & BREAK_BIT) == BREAK_BIT) commandBreak();

    //If the unpair bit was set command an unpair
    if ((ThisEvent.EventParam & UNPAIR_BIT) == UNPAIR_BIT)
    {
        ES_Event ThisEvent;
        ThisEvent.EventType = ES_UNPAIR_REQUEST;
        PostMaster_SM(ThisEvent);
    }

    //check for and maybe send special action
    if ((ThisEvent.EventParam & SPECIAL_ACTION_BIT) ==
        SPECIAL_ACTION_BIT) specialAction();
    ES_Timer_InitTimer(TRANSMIT_TIMER, TRANSMIT_TIME);//restart 1s
                       transmit timeout timer
    sendStatus();//transmit status w pairing bit set
}
//for any upairing event
if ( ThisEvent.EventType == ES_DECRYPT_FAILURE ||ThisEvent.EventType ==
     ES_UNPAIR_REQUEST ||
        (ThisEvent.EventType == ES_TIMEOUT && (ThisEvent.EventParam ==
         PAIRING_TIMER || ThisEvent.EventParam == TRANSMIT_TIMER)) )
{

    //stop the lift fan, disengate brakes, and turn music
    commandCurrentLift(LIFT_OFF);
    stopPlayingSong();
    commandCurrentThrust(0);
    disengageBrakes();

    //Disable all timers and set pair bit to false
    ES_Timer_StopTimer(PAIRING_TIMER);
    ES_Timer_StopTimer(TRANSMIT_TIMER);
    ES_Timer_StopTimer(DMC_TIMING_TIMER);
    is_paired = false;

    if (ThisEvent.EventType == ES_DECRYPT_FAILURE) decryption_failure =
        false;

    //Set updated status, set deactivated bits
    sendStatus();
    setCurrentColor(COLOR_DEACTIVATE);
    setCurrentPairingStatus(!PAIRED_BIT);
    CurrentState = _Unpaired;
}

break;
```

```
    }
    return ReturnEvent;
}

/******************************************************************************
 Function
     QueryTemplateSM
******************************************************************************/
MasterState_t QueryMaster_SM ( void )
{
    return (CurrentState);
}

/******************************************************************************
 private functions
 ******************************************************************************/
//getter function for status
uint8_t getCurrentStatus()
{
    if (is_paired == true)
    {
        status |= PAIRED_BIT;
    }
    else
    {
        status &= ~PAIRED_BIT;
    }
    if (decryption_failure == true)
    {
        status |= DECRYPTION_BIT;
    }
    else
    {
        status &= ~DECRYPTION_BIT;
    }
    //STILL NEED TO DEAL WITH ENCRYPTION FAILURE?
    return status;
}

uint16_t getPartnerAddr()
{
    return PartnerAddr;
}
```