```c
/****************************************************************************
 Module
   Receive_SM.c

 Description
      State Machine for Receiving from XBEE
****************************************************************************/
/*----------------------------- 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 "Receive_SM.h"
#include "Helpers.h"
#include "Definitions.h"
#include "UART_Service.h"
#include "Transmit_Service.h"
#include "Master_SM.h"
#include "Outputs_Service.h"

/*----------------------------- Module Defines ----------------------------*/

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

void processPacket(void);
bool isSamePacket(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 ReceiveState_t CurrentState;

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

//Store the message length
static uint8_t messageLength;
static uint8_t receiveIndex;
static uint8_t currentByte;
static uint8_t    int_cksum = 0;
static uint32_t EncryptionIndex;
static uint8_t ThisPacket[5];
static uint8_t LastPacket[5] = {0 ,0 ,0 ,0, 0};
static uint16_t sourceAddr;
static uint8_t checkSum;
//Create an Array to Hold Data
static uint8_t mostRecentDataArray[ENCR_LENGTH]; //set to the size of the
encryption length as this is the max the protocol will ever allow for

//Create an Array to Hold Encryption Key
static uint8_t EncryptionKey[ENCRYP_LENGTH];
static uint8_t ColorBit;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitReceive_SM
 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
     J. Edward Carryer, 10/23/11, 18:55
****************************************************************************/
bool InitReceive_SM ( uint8_t Priority )
{
    //printf("\n\r InitReceive_SM");

    ES_Event ThisEvent;
    //Initialize MyNumber by reading DMC
    MyPriority = Priority;
    // put us into the Initial PseudoState
    CurrentState = _InitReceive;
    ColorBit = 0;
    // post the initial transition event
    ThisEvent.EventType = ES_INIT;
    if (ES_PostToService( MyPriority, ThisEvent) == true)
    {
        return true;
    }
    else
    {
        return false;
    }
}

/****************************************************************************
 Function
     PostReceive_SM

 Parameters
     EF_Event ThisEvent , the event to post to the queue

 Returns
     boolean False if the Enqueue operation failed, True otherwise
****************************************************************************/
bool PostReceive_SM( ES_Event ThisEvent )
{
    return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunReceive_SM

 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
   J. Edward Carryer, 01/15/12, 15:23
****************************************************************************/
ES_Event RunReceive_SM( ES_Event ThisEvent )
{
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

    //For Debugging print every byte
    if ( ThisEvent.EventType == ES_BYTE_RECEIVED)
    {
        //Store the Byte Locally
        currentByte =  ThisEvent.EventParam;
        //printf("Rx: %x\n\r", currentByte);
    }


    switch ( CurrentState )
    {
    //If We are in the init state
    case _InitReceive :
        if ( ThisEvent.EventType == ES_INIT )// only respond to ES_Init
        {
            CurrentState = _Waiting4Start; //exit the init state
        }
        break;

    case _Waiting4Start :
        if ( ThisEvent.EventType == ES_BYTE_RECEIVED)
        {
            //Check if the Byte has the start delimeter
            if (currentByte == START_DELIMITER)
            {
                //Set the current state equal to Waiting for MSB
                CurrentState = _Waiting4MSB;
                //Reset Checksum
                checkSum = 0; //before exiting
            }
        }
        break;

    case _Waiting4MSB :
        //If We have Received a Byte
        if ( ThisEvent.EventType == ES_BYTE_RECEIVED)
        {
            //Store the Byte Locally
            uint8_t currentByte =  ThisEvent.EventParam;
            //Check if the Byte has the start delimeter
            if (currentByte == ZERO_BYTE)
            {
                //Set the current state equal to Waiting for MSB
                CurrentState = _Waiting4LSB;
            }
        }
        break;

    case _Waiting4LSB :
        //If We have Received a Byte
        if ( ThisEvent.EventType == ES_BYTE_RECEIVED)
        {
            //Store the Byte Locally
            uint8_t currentByte =  ThisEvent.EventParam;

            //No Error Check so simply store the byte as the length and proceed
            to the next state
            CurrentState = _Waiting4API;
            messageLength = currentByte;
        }
        break;

    case _Waiting4API :
        if (currentByte == API_IDENTIFIER_ACK)  //we should never get here in
            new protocol
        {
            CurrentState = _Waiting4Start;
        }
        else if (currentByte == API_IDENTIFIER_RECEIVE)
        {
            CurrentState = _Waiting4DataIncomingPacket;
        }
        else
        {
            CurrentState = _Waiting4Start;
        }
        //Set our Receive Index to 4 as we are now on the 4th byte (started
                                                                    couting from
                                                                    0)
        eiveIndex = RX_INDEX_API+1;
        //Add the Current Byte to the Check Sum
        kSum += currentByte;
        int_cksum = 0;
        break;

    case _Waiting4Data_ResultFromTransmit :
        //Note that we are totally ignoring the Frame ID in this case
        if ((receiveIndex == RXt_INDEX_STATUS) && (currentByte ==
            TX_SUCCESS_BYTE))
        {
            //printf("Transmission Successful\n\r");
            else if (receiveIndex == RXt_INDEX_STATUS)
            {
                //nothing

                //Check if this is our last byte
                have arrived at the checksum
                if (receiveIndex == (messageLength + RX_INDEX_API))
                {
                    //Return to Waiting State
                    CurrentState = _Waiting4Start;
                }
                else
                {
                    checkSum += currentByte;
                    receiveIndex++;
                }
                break;

            case _Waiting4DataIncomingPacket :

                if (receiveIndex == RX_INDEX_SOURCE_MSB) //Check the Source
                    Address and Store it
                {
                    sourceAddr = currentByte;
                    sourceAddr = sourceAddr << 8;
                }
                if (receiveIndex == RX_INDEX_SOURCE_LSB)
                {
                    sourceAddr |= currentByte;
                }
                if (receiveIndex == RX_INDEX_OPTIONS && currentByte ==
                    OPTIONS_INCOMING_ADDRESS_BROADCAST)
                {
                    if (sourceAddr != getPartnerAddr()) CurrentState =
                        _Waiting4Start; //if it's not from partner address, go
                        to waiting for start
                }
                else if (receiveIndex == RX_INDEX_OPTIONS)
                {
                    if ((getCurrentStatus() & PAIRED_BIT) == PAIRED_BIT)
                        CurrentState = _Waiting4Start; //if we're already paired,
                        go to waiting for start
                }

                //If We are starting to receive data lets store it all in an
                array
                if (receiveIndex > RX_INDEX_OPTIONS)
                {
                    mostRecentDataArray[receiveIndex - (RX_INDEX_OPTIONS + 1)] =
                                                        currentByte;
                }
                if (receiveIndex == (messageLength + RX_INDEX_API)) //Check if
                    this is our last byte
                {
                    processPacket();
                    CurrentState = _Waiting4Start;
                }
                else    //If not, add it to our checksum
                {
                    checkSum += currentByte;
                    receiveIndex++;
                }
                break;
            }
            return ReturnEvent;
        }

//function that handles the information and sends the events based on that
function
        acket()
        {

            if (messageLength == CTRL_LENGTH) //if it was a control packet
            {
                //check to see if this is the same packet we just received
                mePacket())
                {
                    ES_Timer_InitTimer(TRANSMIT_TIMER, TRANSMIT_TIME);//reset 1s
                                       timer
                    sendStatus();
                    return;
                }

                for (int i = 0; i < 5; i++) //store the packet in most recent
```

```
                 data
        {
            LastPacket[i] = mostRecentDataArray[i];
            }
            for (int i = 0; i < 5; i++) //decrypt using rolling index
        {
            mostRecentDataArray[i] = mostRecentDataArray[i]^EncryptionKey[
                                     EncryptionIndex%ENCRYP_LENGTH];
                EncryptionIndex++;
                if ( i < 4) int_cksum += mostRecentDataArray[i];
            }
            if ( int_cksum == mostRecentDataArray[CKSUM_INDEX])
            {
                //printf("Intermediate Checksum is successful\r\n");
                //go back to waiting for 7E
                ES_Event ThisEvent;
                ThisEvent.EventType = ES_DECRYPT_FAILURE;//set decrypt
                                      failure and unpair bits then send
                                      status
                PostMaster_SM(ThisEvent);
                CurrentState = _Waiting4Start;
                return;
            }
        }
        if ((checkSum + currentByte == 0xFF))
        {
            ES_Event ThisEvent;//Send Event to MasterSM
            if (mostRecentDataArray[HEADER_INDEX] == PAIR_HDR) //if it was a
                pairing packet
            {
                //check to see if request number is our number
                if ( (mostRecentDataArray[PAIR_BYTE] & PAIRING_NUMBER) ==
                     getCurrentTeamNumber() )
                {
                    ThisEvent.EventType = ES_PAIR_REQUEST;//send pair
                                          request event, store the sender's
                                          address as the param
                    ThisEvent.EventParam = sourceAddr;
                    ColorBit = 0x80 & mostRecentDataArray[1];
                    PostMaster_SM(ThisEvent);
                }
            }
            if (mostRecentDataArray[HEADER_INDEX] == ENCR_HDR) //if it was
                encryption packet
            {
                ThisEvent.EventType = ES_ENCRYPTION_RECVD;
                PostMaster_SM(ThisEvent);
                //store encryption key
                i = 0;
                i < ENCRYP_LENGTH;
                i++)EncryptionKey[i] = mostRecentDataArray[1+i];
            }
            if (mostRecentDataArray[HEADER_INDEX] == CTRL_HDR) //if control
                packet, send control events
            {
                //send control events
                vent.EventType = ES_CTRL_THRUST;
                ThisEvent.EventParam = mostRecentDataArray[THRUST_INDEX];
                PostMaster_SM(ThisEvent);

                ThisEvent.EventType = ES_CTRL_ORIENT;
                ThisEvent.EventParam = mostRecentDataArray[ORIENT_INDEX];
```

```
                    PostMaster_SM(ThisEvent);

                    ThisEvent.EventType = ES_CTRL_SPECIAL;
                    ThisEvent.EventParam = mostRecentDataArray[SPECIAL_INDEX];
                    PostMaster_SM(ThisEvent);
                }
            }
            return;
        }

//for checking agains last packet
        uint8_t getLastEncryptedCksum()
        {
            return LastPacket[4];
        }

        /**************************************************************************
        ****
        QueryTemplateSM

         Parameters
             None

         Returns
             TemplateState_t The current state of the Template state machine

         Description
             returns the current state of the Template state machine
         Notes

         Author
             J. Edward Carryer, 10/23/11, 19:21
        **************************************************************************
        ***/
//getter function for current state
        ReceiveState_t QueryReceive_SM ( void )
        {
            return (CurrentState);
        }

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

         *************************************************************************
         ***/
//to make sure we start at 0
        void setEncryptionIndex()
        {
            EncryptionIndex = 0;
        }

//returns bool after checking against last packet
        acket()
        {
            for (int i = 0; i < 5; i++)
            {
                if (LastPacket[i] != mostRecentDataArray[i]) return false;
            }
            return true;
        }

//getter for received color bit
```

```
8_t getColorBit()
{
    if (ColorBit == 0x80) return 0;
    else return 0x80;
}
```