```
/****************************************************************************
 Module
   Farmer_TX_SM.c

 Revision
   1.0.1

 Description
   The receiving state machine for the Farmer

 Notes

 History
 When           Who     What/Why
 -------------- ---     ---------
 05/13/17 5:29    mwm               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 "FarmerTXSM.h"
#include "Constants.h"
#include "I2C_Service.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"

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

/*---------------------------- Module Functions ---------------------------*/
/* prototypes for private functions for this machine.They should be functions
   relevant to the behavior of this state machine
*/
static void MessageTransmitted( void );
static void ClearMessageArray( void );
static void GenCheckSum( void );
static void BuildPacket(uint8_t packetType);
static void BuildPreamble(void);
static void BuildReq2PairPacket(void);
static void BuildEncrKeyPacket(void);
static void BuildCtrlPacket(void);
static void generateEncryptionKey(void);
static void calculateChecksum(void); //probably don't need this since GenCheckSum
exists
static void setDriveCtrl(void);
static void setSteeringCtrl(void);
static void setDigitalCtrl(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 FarmerTX_State_t CurrentState;
```

```
// with the introduction of Gen2, we need a module level Priority var as well
static uint8_t MyPriority;
static uint8_t MessIndex;
static uint8_t BytesRemaining;
static uint8_t DogTag;
static uint8_t DriveCtrl;
static uint8_t SteeringCtrl;
static uint8_t DigitalCtrl;
static uint8_t DataHeader;
static uint8_t DestAddrMSB;
static uint8_t DestAddrLSB;
static uint8_t PacketLength;
static uint8_t DataLength;
static uint8_t DataIndex;
static uint8_t Checksum;
static bool TransEnable;
static bool ReverseActive;
static bool LeftBrakeActive;
static bool RightBrakeActive;
static bool PeripheralToggled;



static uint8_t Message[TX_MESSAGE_LENGTH] = {0};
static uint8_t EncryptionKey[32];
static uint8_t EncryptionKeyIndex;


/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitFarmerTXSM

 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 InitFarmerTXSM ( uint8_t Priority )
{
  ES_Event ThisEvent;

  MyPriority = Priority;
  // put us into the first state
  CurrentState = Waiting2Transmit;

      //Start TransmitTimer for 200 ms
//    ES_Timer_InitTimer(TRANS_TIMER, TRANSMISSION_RATE);
      //Set Trans_Enable to false
      TransEnable = false; //disable transmission at startup
      ReverseActive = false; // disable reverse at startup
      LeftBrakeActive = false; // disable right brake at startup
      RightBrakeActive = false; // disable left brake at startup
      PeripheralToggled = false; // disable peripheral function at startup
      DriveCtrl = IDLE; // zero thrust fan effort at startup
      SteeringCtrl = STRAIGHT; //no brakes enabled at startup


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

/****************************************************************************
 Function
     PostFarmerTXSM

 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 PostFarmerTXSM( ES_Event ThisEvent )
{
  return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunFarmerTXSM

 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 RunFarmerTXSM( ES_Event ThisEvent )
{
  ES_Event ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

  switch ( CurrentState )
  {
            //Case Waiting2Transmit
            case Waiting2Transmit :
                  //If ThisEvent is ES_TIMEOUT and Transmit is enabled
      /**************************************************************************
************************
                  //if((ThisEvent.EventType == ES_TIMEOUT) &&
(ThisEvent.EventParam == TRANS_TIMER) && TransEnable)

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

                  if(ThisEvent.EventType == ES_SEND_RESPONSE)
                  {
                        //printf("Farmer TX SM -- Waiting2Transmit State --
ES_TIMEOUT and transmit enabled\r\n");
                        //printf("Farmer TX SM -- Waiting2Transmit State --
ES__SEND_RESPONSE\r\n");
                        //Set CurrentState to Transmit
                        CurrentState = Transmit;

                        //Build the message to send
                        BuildPacket(DataHeader);

                        //Reset the message counter (packet byte index)
                        MessIndex = 0;

                        //MAKE SURE DATA LENGTH GETS SET WHEN MESSAGE TYPE GETS SET
                        BytesRemaining = TX_PREAMBLE_LENGTH + DataLength + 1; //
bytes to write = preamble + data + checksum
                        //BytesRemaining = 14;

                        //if TXFE clear
                        if((HWREG(UART1_BASE+UART_O_FR) & UART_FR_TXFE) != 0)
                        {
                              //printf("Farmer TX SM -- Waiting2Transmit State --
Sending Message\r\n");
                              //Write first byte of the message to send into the UART
data register
                              HWREG(UART1_BASE+UART_O_DR) = Message[MessIndex];
                              //decrement BytesRemaining
                              BytesRemaining--;
                              //increment messIndex
                              MessIndex++;
                              //if TXFe clear
                              if((HWREG(UART1_BASE+UART_O_FR) & UART_FR_TXFE) !=
0)
                              {


                                    //Write second byte of the message to send into
the UART data register
                                    HWREG(UART1_BASE+UART_O_DR) =
Message[MessIndex];
                                    //decrement BytesRemaining
                                    BytesRemaining--;
                                    //increment messIndex
                                    MessIndex++;
                              }
                              //Enable Tx interrupts in the UART
                              HWREG(UART1_BASE + UART_O_IM) = HWREG(UART1_BASE +
UART_O_IM) | UART_IM_TXIM;
                        }
                  }


                  break;
```

```
            //Case Transmit
            case Transmit :
                  //If ThisEvent is ES_TRANSMIT_COMPLETE
                  if(ThisEvent.EventType == ES_TRANSMIT_COMPLETE){
                        //printf("Farmer TX SM -- Transmit State -- Transmit
Completed\r\n");
                        //Set CurrentState to Waiting2Transmit
                        CurrentState = Waiting2Transmit;

                        //Set TransEnable to false
                        //TransEnable = false;
                        MessageTransmitted();
                  }
                  break;
    default :
      ;
  }  // end switch on Current State
  return ReturnEvent;
}

/****************************************************************************
 Function
     QueryFarmerTXSM

 Parameters
     None

 Returns
     FarmerTX_State_t The current state of the Template state machine

 Description
     returns the current state of the Template state machine
 Notes

 Author
Matthew Miller, 5/13/17, 22:42
****************************************************************************/
FarmerTX_State_t QueryFarmerTXSM ( void )
{
   return(CurrentState);
}
/****************************************************************************
 Function
     FarmerTX_ISR

 Parameters
     None

 Returns
     The interrupt response for the UART receive

 Description
     stores the received byte into the data
 Notes

 Author
Matthew Miller, 5/13/17, 22:42
****************************************************************************/
void FarmerTX_ISR( void ){
      //Write next byte of message
      HWREG(UART1_BASE+UART_O_DR) = Message[MessIndex];

      //Decrement BytesRemaining
      BytesRemaining--;
      //Increment messIndex
      MessIndex++;
      if(MessIndex > 42)
      {
            //printf("FATAL TRANSMIT OVERFLOW ERROR\r\n");
      }
      //If BytesRemaining is 0
      if(BytesRemaining == 0){
            //Disable interrupt on TX
            HWREG(UART1_BASE + UART_O_IM) = HWREG(UART1_BASE + UART_O_IM) &
~UART_IM_TXIM;

            //Post ES_TRANSMIT_COMPLETE event
            ES_Event ReturnEvent;
            ReturnEvent.EventType = ES_TRANSMIT_COMPLETE;
            PostFarmerTXSM(ReturnEvent);
      }
}

void enableTransmit( void ){
      TransEnable = true;
      return;
}

void disableTransmit(void)
{
      TransEnable = false;
}


//Sets the DataHeader to the correct message type and updates the length of the data
void setFarmerDataHeader(uint8_t Header)
{
      //Set DataHeader to Header
      DataHeader = Header;

      //if DataHeader is REQ_2_PAIR
      if(DataHeader == REQ_2_PAIR)
      {
            //printf("Farmer TX SM -- Data Header -- REQ_2_PAIR\r\n");
            //Set DataLength to REQ_2_PAIR_LENGTH
            DataLength = REQ_2_PAIR_LENGTH;
      }
      //ElseIf DataHeader is ENCR_KEY
      else if(DataHeader == ENCR_KEY)
      {
            //printf("Farmer TX SM -- Data Header -- ENCR_KEY\r\n");
            //Set DataLength to ENCR_KEY_LENGTH
            DataLength = ENCR_KEY_LENGTH;
      }
      //ElseIf DataHeader is CTRL
      else if (DataHeader ==  CTRL)
      {
            //printf("Farmer TX SM -- Data Header -- CTRL\r\n");
            //Set DataLength to CTRL_LENGTH
            DataLength = CTRL_LENGTH;
      }//EndIf
      else //must be an unintended message type
      {
            //print an error message
            //printf("FARMER DATAHEADER SET TO UNEXPECTED MESSAGE TYPE");
      }
}
```

```c
//Sets the Destination XBEE address the message will be sent to
void setDestDogAddress(uint8_t AddrMSB, uint8_t AddrLSB)
{
      //printf("Set Destination Dog Address -- ADDRESS\r\n");
      //Set Destination MSB to AddrMSB
      DestAddrMSB = AddrMSB;
      //DestAddrMSB = 0x21;
      //Set Destination LSB to AddrLSB
      DestAddrLSB = AddrLSB;
      //DestAddrLSB = 0x81;
}

//Sets the DogTag number of the Dog to be paired with from a REQ_2_PAIR command
void setDogTag(uint8_t TagNumber)
{
      //Set DogTag to TagNumber
      DogTag = TagNumber;
}

void EnableReverse(void)
{
      //Set reverse flag
      ReverseActive = true;
}

void DisableReverse(void)
{
      //Clear reverse flag
      ReverseActive = false;
}

// Turn on left brake
void EnableLeftBrake(void)
{
      LeftBrakeActive = true;
}

// Turn off left brake
void DisableLeftBrake(void)
{
      LeftBrakeActive = false;
}

// Turn on right brake
void EnableRightBrake(void)
{
      RightBrakeActive = true;
}

// Turn off right brake
void DisableRightBrake(void)
{
      RightBrakeActive = false;
}

void TogglePeripheral(void)
{
      PeripheralToggled = !PeripheralToggled;
}

uint8_t getDestAddrMSB(void)
{
      return DestAddrMSB;
```

```
}

uint8_t getDestAddrLSB(void)
{
      return DestAddrLSB;
}

void resetEncryptionIndex(void)
{
      EncryptionKeyIndex = 0;
}

uint8_t getEncryptionKeyIndex(void)
{
      return EncryptionKeyIndex;
}

void clearControls(void)
{
      RightBrakeActive = false;
      LeftBrakeActive = false;
      ReverseActive = false;
      PeripheralToggled = false;
}


/***************************************************************************
 private functions
 ***************************************************************************/
static void MessageTransmitted()
{

      //printf("Packet length: %i bytes\r\n", TX_PREAMBLE_LENGTH+DataLength+1);

      /*
      for(int i = 0; i<(TX_PREAMBLE_LENGTH+DataLength+1);i++)
      {
            printf("TX %i: %04x\r\n",i,Message[i]);
      }
      */

      return;

}



static void ClearMessageArray( void )
{
      for(int i = 0; i<(TX_PREAMBLE_LENGTH+DataLength+1);i++)
      {
            Message[i] = 0;
      }
      return;
}


static void BuildPacket(uint8_t packetType)
{
      //printf("Build Packet -- TOP\r\n");
            //Build the preamble of the packet
            BuildPreamble();
            //If packetType is REQ_2_PAIR
            if(packetType == REQ_2_PAIR)
```

```
        {
            //printf("Build Packet -- BuildPacket -- REQ2PAIR\r\n");
            //Build the rest of the data as a REQ_2_PAIR packet
            BuildReq2PairPacket();
        }
        //Else If packetType is ENCR_KEY
        else if(packetType == ENCR_KEY)
        {
            //printf("Build Packet -- BuildPacket -- ENCR_KEY\r\n");
            //Build the rest of the data as an ENCR_KEY packetType
            BuildEncrKeyPacket();
        }
        //Else If packetType is CTRL
        else if(packetType == CTRL)
        {
            //printf("Build Packet -- BuildPacket -- CTRL\r\n");
            //Build the rest of the data as a CTRL packet
            BuildCtrlPacket();
        }
        else //     Else we must have gotten an unexpected packet type
        {
            //Print an error message to show we got a bad packet request
            //printf("UNEXPECTED PACKET TYPE REQUESTED TO TRANSMIT");
        }
//    EndIf
}


static void BuildPreamble(void)
{
    //Store START_DELIMITER in byte 0 of PacketArray
    Message[0] = START_DELIMITER;
    //Store PACKET_LENGTH_MSB in byte 1 of PacketArray (0x00)
    Message[1] = PACKET_LENGTH_MSB;
    //Store DataLength in byte 2 of PacketArray
    Message[2] = DataLength + FRAME_DATA_PREAMBLE_LENGTH;
    //Store TX_API_IDENTIFIER in byte 3 of PacketArray (0x01)
    Message[3] = TX_API_IDENTIFIER;
    //Store TX_FRAME_ID in byte 4 of PacketArray (Should this be 0x00 or a different
value?)
    Message[4] = TX_FRAME_ID;
    //Store DestAddrMSB in byte 5 of PacketArray (Write 0xff to both for broadcast)
    Message[5] = DestAddrMSB;
    //Store DestAddrLSB in byte 6 of PacketArray (Write 0xff to both for broadcast)
    Message[6] = DestAddrLSB;
    //Store OPTIONS in byte 7 of PacketArray (0x00)
    Message[7] = OPTIONS;
}


static void BuildReq2PairPacket(void)
{
    //printf("Build Packet -- Building the Packet -- REQ2PAIR\r\n");
    //Set DataIndex to TX_PREAMBLE_LENGTH
    DataIndex = TX_PREAMBLE_LENGTH;
    //Store DataHeader in byte DataIndex of PacketArray
    Message[DataIndex] = DataHeader;

    //Increment DataIndex
    DataIndex++;
    //Store DogTag in byte DataIndex of PacketArray
    Message[DataIndex] = DogTag;

    //Increment DataIndex
```

```
      DataIndex++;
      //Calculate the checksum
      calculateChecksum();
      //Store the checksum in byte DataIndex of PacketArray
      Message[DataIndex] = Checksum;
}


static void BuildEncrKeyPacket(void)
{
      //printf("Build Packet -- Building the Packet -- Encr\r\n");
      //Set DataIndex to TX_PREAMBLE_LENGTH
      DataIndex = TX_PREAMBLE_LENGTH;
      //Store DataHeader in byte DataIndex of PacketArray
      Message[DataIndex] = DataHeader;

      //Generate a new encyption key since we are attempting a new pair
      generateEncryptionKey();

      //Loop ENCR_KEY_LENGTH - 1 times (we don't include the header)
      for(uint8_t i = 0; i < ENCR_KEY_LENGTH-1; i++)
      {
            //Increment DataIndex
            DataIndex++;
            //Store element i of EncryptionKey in byte DataIndex of PacketArray
            Message[DataIndex] = EncryptionKey[i];
      }//EndLoop

      //Reset EncryptionKeyIndex
      EncryptionKeyIndex = 0;

      //Increment DataIndex
      DataIndex++;
      //Calculate the checksum
      calculateChecksum();
      //Store the checksum in byte DataIndex of PacketArray
      Message[DataIndex] = Checksum;
}


static void BuildCtrlPacket(void)
{

      //Set DataIndex to TX_PREAMBLE_LENGTH
      DataIndex = TX_PREAMBLE_LENGTH;
      //Encrypt DataHeader using element of EncryptionKey corresponding to
EncryptionKeyIndex and store in Messaage
      //printf("Unencrypted Byte: %i, EncryptionKeyIndex: %i, EncryptionKey:
%i\r\n", DataHeader, EncryptionKeyIndex, EncryptionKey[EncryptionKeyIndex]);
      Message[DataIndex] = DataHeader^ EncryptionKey[EncryptionKeyIndex];
      //Increment EncryptionKeyIndex (modulo 32)
      EncryptionKeyIndex = (EncryptionKeyIndex + 1)%32;


      //Increment DataIndex
      DataIndex++;

      //Set the DriveCtrl value based on the accelerometer reading and the Reverse
button
      setDriveCtrl();

      //Encrypt DriveCtrl using element of EncryptionKey corresponding to
EncryptionKeyIndex and store in Message
      //printf("Unencrypted Byte: %i, EncryptionKeyIndex: %i, EncryptionKey:
```

```
%i\r\n", DriveCtrl, EncryptionKeyIndex, EncryptionKey[EncryptionKeyIndex]);
      Message[DataIndex] = DriveCtrl^ EncryptionKey[EncryptionKeyIndex];
      //Increment EncryptionKeyIndex (modulo 32)
      EncryptionKeyIndex = (EncryptionKeyIndex + 1)%32;

      //Increment DataIndex
      DataIndex++;

      //Set the SteeringCtrl value based on the state of the brake buttons
      setSteeringCtrl();

      //Encrypt SteeringCtrl using element of EncryptionKey corresponding to
EncryptionKeyIndex and Store in Message
      //printf("Unencrypted Byte: %i, EncryptionKeyIndex: %i, EncryptionKey:
%i\r\n", SteeringCtrl, EncryptionKeyIndex, EncryptionKey[EncryptionKeyIndex]);
      Message[DataIndex] = SteeringCtrl^ EncryptionKey[EncryptionKeyIndex];
      //Increment EncryptionKeyIndex (modulo 32)
      EncryptionKeyIndex = (EncryptionKeyIndex + 1)%32;


      //Increment DataIndex
      DataIndex++;

      //Set the DigitalCtrl value based on the state of the peripheral/brake buttons
      setDigitalCtrl();

      //Encrypt DigitalCtrl using element of EncryptionKey corresponding to
EncryptionKeyIndex and store in Message
      //printf("Unencrypted Byte: %i, EncryptionKeyIndex: %i, EncryptionKey:
%i\r\n", DigitalCtrl, EncryptionKeyIndex, EncryptionKey[EncryptionKeyIndex]);
      Message[DataIndex] = DigitalCtrl^ EncryptionKey[EncryptionKeyIndex];
      //Increment EncryptionKeyIndex (modulo 32)
      EncryptionKeyIndex = (EncryptionKeyIndex + 1)%32;

      //Increment dataIndex
      DataIndex++;
      //Calculate the checksum
      calculateChecksum();
      //Store the checksum in byte dataIndex of PacketArray
      Message[DataIndex] = Checksum;
}


static void generateEncryptionKey(void)
{
      //Loop ENCR_KEY_LENGTH - 1 times (we don't want to count the header)
      for(uint8_t i = 0; i < ENCR_KEY_LENGTH-1; i++)
      {
            //Generate a random 8 bit number and store in EncryptionKey array
            EncryptionKey[i] = rand()%256;
      }//EndLoop
}



static void calculateChecksum(void) //probably don't need this since GenCheckSum
exists
{
      //local variable Sum
      uint8_t Sum;
      //local variable Index
      uint8_t Index;
      //local variable FrameDataLength
      uint8_t     FrameDataLength;
      //Initialize Sum to 0
      Sum = 0;

      //Set FrameDataLength to DataLength + FRAME_DATA_PREAMBLE_LENGTH (5)
      FrameDataLength = DataLength + FRAME_DATA_PREAMBLE_LENGTH;

      //Loop FrameDataLength times
      //start Index at 3 (where the frame data begins_
      for(Index = FRAME_DATA_START; Index < FRAME_DATA_START + FrameDataLength;
Index++)
      {
            //Add element Index of PacketArray to Sum
            Sum += Message[Index];
            //printf("Current Sum: %i\r\n", Sum);
      }//End Loop

      //Subtract Sum from 0xff and store in Checksum
      Checksum = 0xFF - Sum;
}


//Sets the Drive Control byte for a control message
static void setDriveCtrl(void)
{
      //scale the speed of the thrust fan based upon the period measured in the IMU
      uint16_t Period;
      Period = getPeriod();

      //if we want to go forward
      if(!ReverseActive)
      {
            //if the period is faster than 300, saturate the value
            if(Period < 300)
            {
                  DriveCtrl = MAX_FORWARD;
            }
            //else scale the value between 127 and 255
            else
            {
                  DriveCtrl = (uint8_t)((((1000-(uint32_t)Period)*128)/700)+127);

            }
      }
      //else we want to go in reverse
      else
      {
            //if the period is faster than 300, saturate the value
            if(Period < 300)
            {
                  DriveCtrl = MAX_REVERSE;
            }
            //else scale the value between 127 and 0
            else
            {
                  DriveCtrl = (uint8_t)(127-(((1000-(uint32_t)Period)*127)/700));
            }
      }
      printf("THRUST FAN DUTY CYCLE = %i \r\n", DriveCtrl);
}


//Sets the Steering Control byte for a control message
static void setSteeringCtrl(void)
```

```
{
      //Set SteeringCtrl to CtrlByte
      //SteeringCtrl = CtrlByte;

      //If both brakes are active, we want to enable the brake bit and still go
straight
      if(RightBrakeActive && LeftBrakeActive)
      {
            SteeringCtrl = STRAIGHT;
      }
      //If we are only braking right we want to set the steering to be all the way
right
      else if(RightBrakeActive)
      {
            SteeringCtrl = MAX_RIGHT_TURN;
      }
      //If we are only braking left we want to set the steering to be all the way
left
      else if(LeftBrakeActive)
      {
            SteeringCtrl = MAX_LEFT_TURN;
      }
      //If no brakes are active we want to go straight
      else
      {
            SteeringCtrl = STRAIGHT;
      }
}

//Sets the Digital Control byte for a control message
static void setDigitalCtrl(void)
{
      //Set DigitalCtrl to CtrlByte
      //DigitalCtrl = CtrlByte;

      //If the peripheral is set
      if(PeripheralToggled)
      {
            //Set the peripheral bit in DigitalCtrl
            DigitalCtrl |= BIT0HI;
            //only set it for this one data packet
            PeripheralToggled = false;
      }//EndIf

      else //we want to make sure to send a 0 to the DOG
      {
            DigitalCtrl &= ~(BIT0HI);
      }

      //If both Left and Right brakes are active
      if(LeftBrakeActive && RightBrakeActive)
      {
            //Set the braking bit in DigitalCtrl
            DigitalCtrl |= BIT1HI;
      }//EndIf

      else //make sure the braking bit is CLEAR
      {
            DigitalCtrl &= ~(BIT1HI);
      }


}
```