```
/*******************************************************************************
 Module
   Dog_TX_SM.c

 Revision
   1.0.1

 Description
   The receiving state machine for the Dog

 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 "DogTXSM.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 BuildPacket(uint8_t packetType);
static void BuildPreamble(void);
static void BuildPairAck(void);
static void BuildEncrReset(void);
static void BuildStatus(void);
static void calculateChecksum(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 DogTX_State_t CurrentState;

// with the introduction of Gen2, we need a module level Priority var as well
static uint8_t MyPriority, MessIndex, BytesRemaining;
static uint8_t DataLength;
static uint8_t DataHeader;
static uint8_t DestAddrMSB;
static uint8_t DestAddrLSB;
```

```
static uint8_t DataIndex;
static uint8_t Message[TX_MESSAGE_LENGTH] = {0};
static uint8_t Checksum;

//IMU data
static uint8_t AccelX_MSB;
static uint8_t AccelX_LSB;
static uint8_t AccelY_MSB;
static uint8_t AccelY_LSB;
static uint8_t AccelZ_MSB;
static uint8_t AccelZ_LSB;

static uint8_t GyroX_MSB;
static uint8_t GyroX_LSB;
static uint8_t GyroY_MSB;
static uint8_t GyroY_LSB;
static uint8_t GyroZ_MSB;
static uint8_t GyroZ_LSB;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitDogTXSM

 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 InitDogTXSM ( uint8_t Priority )
{
  ES_Event ThisEvent;

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


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

/****************************************************************************
 Function
     PostDogTXSM

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

/****************************************************************************
 Function
    RunDogTXSM

 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 RunDogTXSM( ES_Event ThisEvent )
{
  ES_Event ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
      HWREG(GPIO_PORTB_BASE + ALL_BITS) ^= (GPIO_PIN_1);

  switch ( CurrentState )
  {
            //Case Waiting2Transmit
            case Waiting2Transmit :
                  //If ThisEvent is ES_SEND_RESPONSE then we want to send something
back to the Farmer
                  if(ThisEvent.EventType == ES_SEND_RESPONSE)
                  {
                        printf("Dog TX SM -- Waiting2Transmit State -- Send Request
Received\r\n");
                        //Set CurrentState to Transmit
                        CurrentState = Transmit;
                        //Build the message to send
                        BuildPacket(DataHeader);
                        //Reset the message counter (packet byte index)
                        MessIndex = 0;
                        BytesRemaining = TX_PREAMBLE_LENGTH + DataLength + 1;
//length of message is preamble + data + checksum
                        //BytesRemaining = 14;
                        //if TXFE clear
                        if((HWREG(UART1_BASE+UART_O_FR) & UART_FR_TXFE) != 0)
                        {
                              //printf("Dog TX SM -- Waiting2Transmit State --
First\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)
                                {
                                      //printf("Dog TX SM -- Waiting2Transmit State
-- Second\r\n");
                                      //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("Dog TX SM -- Transmit State -- TRANSMIT
COMPLETE\r\n");
                          //Set CurrentState to Waiting2Transmit
                          CurrentState = Waiting2Transmit;
                          MessageTransmitted();
                    }
                    break;
     default :
       ;
   }  // end switch on Current State
   return ReturnEvent;
}

/****************************************************************************
 Function
     QueryDogTXSM

 Parameters
     None

 Returns
     DogTX_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
****************************************************************************/
DogTX_State_t QueryDogTXSM ( void )
{
```

```
   return(CurrentState);
}
/****************************************************************************
 Function
     DogTX_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 DogTX_ISR( void ){
      //printf(".");
      //Write next byte of message
      HWREG(UART1_BASE+UART_O_DR) = Message[MessIndex];

      //Decrement BytesRemaining
      BytesRemaining--;

      //Increment messIndex
      MessIndex++;

      //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;
            PostDogTXSM(ReturnEvent);
      }
}

void setDogDataHeader(uint8_t Header)
{
      //Set DataHeader to Header
      DataHeader = Header;

      //If DataHeader is PAIR_ACK
      if(DataHeader == PAIR_ACK)
      {
            printf("Dog TX SM -- Set Data Header -- PAIR_ACK\r\n");
            //Set the data length to PAIR_ACK_LENGTH
            DataLength = PAIR_ACK_LENGTH;
      }
      //ElseIf DataHeader is ENCR_RESET
      else if(DataHeader == ENCR_RESET)
      {
            printf("Dog TX SM -- Set Data Header -- ENCR_RESET\r\n");
            //Set the data length to ENCR_RESET_LENGTH
            DataLength = ENCR_RESET_LENGTH;
      }
      //ElseIf DataHeader is STATUS
      else if(DataHeader == STATUS)
```

```
	{
		printf("Dog TX SM -- Set Data Header -- STATUS\r\n");
		//Set the data length to STATUS_LENGTH
		DataLength = STATUS_LENGTH;
	}
	else
	{
		//Data header is of unexpected type, print an error message
		printf("DOG DATAHEADER SET TO UNEXPECTED MESSAGE TYPE");
	}//EndIf
}

void setDestFarmerAddress(uint8_t AddrMSB, uint8_t AddrLSB)
{
	//Set Destination MSB to AddrMSB
	DestAddrMSB = AddrMSB;
	//Set Destination LSB to AddrLSB
	DestAddrLSB = AddrLSB;
}

uint8_t getDestFarmerAddressLSB( void ){
	return DestAddrLSB;
}

uint8_t getDestFarmerAddressMSB( void ){
	return DestAddrMSB;
}

/*****************************************************************************
 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 BuildPacket(uint8_t packetType)
{
		//Build the preamble of the packet
		BuildPreamble();
		//If packetType is PAIR_ACK
		if(packetType == PAIR_ACK)
		{
			//Build the rest of the data as a REQ_2_PAIR packet
			BuildPairAck();
		}
		//Else If packetType is ENCR_RESET
		else if(packetType == ENCR_RESET)
		{
			//Build the rest of the data as an ENCR_RESET packetType
			BuildEncrReset();
		}
		//Else If packetType is CTRL
		else if(packetType == STATUS)
		{
			//Build the rest of the data as a status packet
			BuildStatus();
		}
```

```
            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;
      //Message[5] = 0x20;
      //Store DestAddrLSB in byte 6 of PacketArray (Write 0xff to both for broadcast)
      Message[6] = DestAddrLSB;
      //Message[6] = 0x81;
      //Store OPTIONS in byte 7 of PacketArray (0x00)
      Message[7] = OPTIONS;
}


static void BuildPairAck(void)
{
      //Set DataIndex to first byte of RF_data (byte 9)
      DataIndex = TX_PREAMBLE_LENGTH;
      //Set the 9th byte of Message to the data header
      Message[DataIndex] = DataHeader;

      //Increment DataIndex
      DataIndex++;
      //Calculate the Checksum
      calculateChecksum();
      //store the checksum in the message array
      Message[DataIndex] = Checksum;
}


static void BuildEncrReset(void)
{
      //Set DataIndex to first byte of RF_data (byte 9)
      DataIndex = TX_PREAMBLE_LENGTH;
      //Set the 9th byte of Message to the data header
      Message[DataIndex] = DataHeader;

      //Increment DataIndex
      DataIndex++;
      //Calculate the Checksum
      calculateChecksum();
      //store the checksum in the message array
      Message[DataIndex] = Checksum;
}
```

```
static void BuildStatus(void)
{
	//Set DataIndex to first byte of RF_data (byte 9)
	DataIndex = TX_PREAMBLE_LENGTH;
	//Set the 9th byte of Message to the data header
	Message[DataIndex] = DataHeader;

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getAccelX_MSB(); //Message[DataIndex] =
0x01;//getAccelX_MSB();

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getAccelX_LSB(); //Message[DataIndex] =
0x02;//getAccelX_LSB();

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getAccelY_MSB(); //Message[DataIndex] =
0x03;//getAccelY_MSB();

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getAccelY_LSB(); //Message[DataIndex] =
0x04;//getAccelY_LSB();

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getAccelZ_MSB(); //Message[DataIndex] =
0x05;//getAccelZ_MSB();

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getAccelZ_LSB(); //Message[DataIndex] =
0x06;//getAccelZ_LSB();

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getGyroX_MSB();//Message[DataIndex] =
0x07;//getGyroX_MSB();

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getGyroX_LSB(); //Message[DataIndex] =
0x08;//getGyroX_LSB();

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getGyroY_MSB(); //Message[DataIndex] =
0x09;//getGyroY_MSB();

	//increment DataIndex
	DataIndex++;
```

```
	//Write next IMU byte to message
	Message[DataIndex] = getGyroY_LSB(); //Message[DataIndex] =
0x0A;//getGyroY_LSB();

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getGyroZ_MSB(); //Message[DataIndex] =
0x0B;//getGyroZ_MSB();

	//increment DataIndex
	DataIndex++;
	//Write next IMU byte to message
	Message[DataIndex] = getGyroZ_LSB(); //Message[DataIndex] = 0x0C;
//getGyroZ_LSB();

	//Increment DataIndex
	DataIndex++;
	//Calculate the Checksum
	calculateChecksum();
	//store the checksum in the message array
	Message[DataIndex] = Checksum;
}


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("CurrentSum = %i\r\n",Sum);
	}//End Loop

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