```
/****************************************************************************
 Module
   Dog_RX_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 "DogRXSM.h"
#include "Constants.h"
#include "DogTXSM.h"
#include "DogMasterSM.h"
#include "EventCheckers.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 DataInterpreter( void );
//static void setPair( void );
//static void LostConnection( void );
static void ClearDataBufferArray( void );
static void MoveDataFromBuffer( void );
static void StoreData( 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 DogRX_State_t CurrentState, ISRState;

// with the introduction of Gen2, we need a module level Priority var as well
static uint8_t MyPriority, memCnt, TurnData, MoveData, PerData, BrakeData;
static uint8_t MSB_Address, LSB_Address, EncryptCnt, RecDogTag, Header, Frame_API;
static uint16_t BytesLeft,DataLength,TotalBytes;
static uint8_t Data[RX_MESSAGE_LENGTH] = {0};
static uint8_t DataBuffer[RX_MESSAGE_LENGTH] = {0};
static uint8_t Encryption[ENCR_LENGTH] = {0};
static uint8_t CheckSum;


/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitDogRXSM

 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 InitDogRXSM ( uint8_t Priority )
{
  ES_Event ThisEvent;

  MyPriority = Priority;
  // put us into the first state
  CurrentState = Waiting2Rec;
      ISRState = WaitForFirstByte;
  // post the initial transition event
      //Set memCnt to 0
      memCnt = 0;

      // connect clock to ports B
      HWREG(SYSCTL_RCGCGPIO) |= (SYSCTL_RCGCGPIO_R1);
      // wait for clock to connect to ports B and F
      while ((HWREG(SYSCTL_PRGPIO) & (SYSCTL_PRGPIO_R1)) != (SYSCTL_PRGPIO_R1)) {}
      // digitally enable IO pins
      HWREG(GPIO_PORTB_BASE + GPIO_O_DEN) |= (GPIO_PIN_1);
      // set direction of IO pins
      HWREG(GPIO_PORTB_BASE + GPIO_O_DIR) |= (GPIO_PIN_1);

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

/****************************************************************************
 Function
     PostDogRXSM

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

/****************************************************************************
 Function
    RunDogRXSM

 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 RunDogRXSM( ES_Event ThisEvent )
{
  ES_Event ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
  switch ( CurrentState )
  {
            case Waiting2Rec :
                        //if ThisEvent EventType is ES_BYTE RECEIVED
                        if(ThisEvent.EventType == ES_BYTE_RECEIVED){
                              //Set CurrentState to Receive
                              CurrentState = Receive;
                        }
                  break;

            case Receive :

                  //Handle LOST_CONNECTION_EVENTS
                  if(ThisEvent.EventType == ES_LOST_CONNECTION)
                  {
                              //Set CurrentState to Waiting2Rec
                              CurrentState = Waiting2Rec;
                              //Set memCnt to 0
                              memCnt = 0;
                              //Reset ISRState
                              ISRState = WaitForFirstByte;
                              //Clear Data Array
                              ClearDataArray();
                              //Clear Data Buffer
                              ClearDataBufferArray();
                  }
                  //if ThisEvent EventType is ES_MESSAGE_REC
                  else if(ThisEvent.EventType == ES_MESSAGE_REC){
                        //Turn off timer
                        //Call Data Interpreter
                        DataInterpreter();
                        //Post ES_MESSAGE_REC to DogMasterSM
                        ES_Event NewEvent;
                        NewEvent.EventType = ES_MESSAGE_REC;
                        PostDogMasterSM(NewEvent);

                        //Set CurrentState to Waiting2Rec
                        CurrentState = Waiting2Rec;
                  }
                  break;
    default :
      ;
  }  // end switch on Current State
  return ReturnEvent;

}

/****************************************************************************
 Function
     QueryDogRXSM

 Parameters
     None

 Returns
     DogRX_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
****************************************************************************/
DogRX_State_t QueryDogRXSM ( void )
{
   return(CurrentState);
}
/****************************************************************************
 Function
     DogRX_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 DogRX_ISR( void ){
      ES_Event ReturnEvent;
      //Set data to the current value on the data register
      if(memCnt > 42)
      {
            printf("FATAL ARRAY OVERFLOW ERROR: %i\r\n", memCnt);
      }

      DataBuffer[memCnt] = HWREG(UART1_BASE + UART_O_DR);
      //Check and handle receive errors
      if((HWREG(UART1_BASE + UART_O_RSR) & UART_RSR_OE) != 0){
            printf("Overrun Error :(\r\n");
      }
      if((HWREG(UART1_BASE + UART_O_RSR) & UART_RSR_BE) != 0){
            printf("Break Error :(\r\n");
      }
      if((HWREG(UART1_BASE + UART_O_RSR) & UART_RSR_FE) != 0){
            printf("Framing Error :(\r\n");
      }
      if((HWREG(UART1_BASE + UART_O_RSR) & UART_RSR_PE) != 0){
            printf("Parity Error :(\r\n");
      }
      HWREG(UART1_BASE + UART_O_ECR) |= UART_ECR_DATA_M;
  switch ( ISRState )
  {
       //Case WaitForFirstByte
            case WaitForFirstByte:
            if(DataBuffer[0] == INIT_BYTE)
            {
                  HWREG(GPIO_PORTB_BASE + ALL_BITS) |= BIT1HI;
                  //Set ISRState to WaitForMSBLen
                  ISRState = WaitForMSBLen;
                  //Increment memCnt
                  memCnt++;

                  //Post ES_BYTE_RECEIVED event to  FarmerRXSM
                  ReturnEvent.EventType = ES_BYTE_RECEIVED;
                  PostDogRXSM(ReturnEvent);
            }
            break;

            //Case WaitForMSBLen
            case WaitForMSBLen :
                  //Set IsRState to WaitForLSBLen
                  ISRState = WaitForLSBLen;
                  //Increment memCnt
                  memCnt++;

            break;

            //Case WaitForLSBLen
            case WaitForLSBLen :
                  //Set ISRState to AcquireData
                  ISRState = AcquireData;

                  //initialize checksum
                  CheckSum = 0;

                  //Increment memCnt
                  memCnt++;

                  //Combine Data[1] and Data[2] into BytesLeft and DataLength
                  BytesLeft = DataBuffer[1];
                  BytesLeft = (BytesLeft << 8) + DataBuffer[2];
                  //printf("Bytes Left Initial value = %i\r\n", BytesLeft);
                  DataLength = BytesLeft;
                  TotalBytes = DataLength+NUM_XBEE_BYTES;

                  break;

            //Case AcquireData
```

```
        case AcquireData :
            if(BytesLeft !=0)
            {
                //Increment memCnt
                CheckSum += DataBuffer[memCnt];
                memCnt++;

                //Decrement BytesLeft
                BytesLeft--;
            }
            else if(BytesLeft == 0)
            {
                CheckSum = 0xff - CheckSum;

                //Set ISRState to WaitForFirstByte
                ISRState = WaitForFirstByte;




                // Only post if it is actual message Dog needs to handle
                if((DataBuffer[3] == API_81) && (CheckSum ==
DataBuffer[memCnt]))
                {
                    ReturnEvent.EventType = ES_MESSAGE_REC;
                    PostDogRXSM(ReturnEvent);
                }
                else if(CheckSum != DataBuffer[memCnt])

                {
                    SetBadCheckSum();
                }

                //Set memCnt to 0
                memCnt = 0;

                //Move and clear DataBuffer
                MoveDataFromBuffer();
                //ClearDataBufferArray();
                HWREG(GPIO_PORTB_BASE + ALL_BITS) &= BIT1LO;
            }
            break;

        default:
            break;

    }
}

void RXTX_ISR( void ){
    //get status of the receive and transmit interrupts
    uint8_t RX_Int = HWREG(UART1_BASE + UART_O_MIS) & UART_MIS_RXMIS;
    uint8_t TX_Int = HWREG(UART1_BASE + UART_O_MIS) & UART_MIS_TXMIS;

    //If there was a receive interrupt
    if(RX_Int != 0){
        //Clear the source of the interrupt
        HWREG(UART1_BASE + UART_O_ICR) |= UART_ICR_RXIC;
        //Call the Dog receive interrupt response
        DogRX_ISR();
    }

    //If there was a transmit interrupt
    if(TX_Int != 0){
            //Clear the source of the interrupt
            HWREG(UART1_BASE + UART_O_ICR) |= UART_ICR_TXIC;
            //Call the Dog transmit interrupt response
            DogTX_ISR();
      }
}



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

      /*
      printf("Dog RX SM -- Data Interpreter -- Top\r\n");
      for(int i = 0; i<TotalBytes;i++){
            printf("RX %i: %04x\r\n",i,Data[i]);
      }
      */
      // Store the data for use by the MasterSM
      StoreData();

}

void ClearDataArray( void ){
      for(int i = 0; i<RX_MESSAGE_LENGTH;i++){
            Data[i] = 0;
      }
}

static void ClearDataBufferArray( void ){
      for(int i = 0; i<RX_MESSAGE_LENGTH;i++){
            DataBuffer[i] = 0;
      }
}

static void MoveDataFromBuffer( void ){
      for(int i = 0; i<RX_MESSAGE_LENGTH;i++){
            Data[i] = DataBuffer[i];
      }
}

static void StoreData( void ){
      //Set Header
      Header = Data[8];
      printf("Dog RX SM -- Store Data -- Header = 0x%04x\r\n", Header);

      //Set API Header
      Frame_API = Data[3];
      printf("Dog RX SM -- Store Data -- Frame API = 0x%04x\r\n", Frame_API);

      //Set MSB_Address
      MSB_Address = Data[4];
      printf("Dog RX SM -- Store Data -- MSB_Address = 0x%04x\r\n", MSB_Address);

      //Set LSB_Address
      LSB_Address = Data[5];
      printf("Dog RX SM -- Store Data -- LSB_Address = 0x%04x\r\n", LSB_Address);

      //Set TurnData
      TurnData = Data[10];
      printf("Dog RX SM -- Store Data -- TurnData = 0x%04x\r\n", TurnData);
      //Set MoveData
      MoveData = Data[9];
      printf("Dog RX SM -- Store Data -- MoveData = 0x%04x\r\n", MoveData);

      //Set Brake
      BrakeData = Data[11] & BRAKE_MASK;
      printf("Dog RX SM -- Store Data -- BrakeData = 0x%04x\r\n", BrakeData);

      //Set Peripheral
      PerData = Data[11] & PER_MASK;
      printf("Dog RX SM -- Store Data -- Peripheral = 0x%04x\r\n", PerData);

      //Set Received DogTag
      RecDogTag = Data[9];
      printf("Dog RX SM -- Store Data -- RecDogTag = 0x%04x\r\n", RecDogTag);
}

void DecryptData( void ){
      printf("Dog RX SM -- Data -- Top\r\n");
      //for each of the elements of the dataBuffer
      // set data equal to dataBuffor xor with Encryption Key

      printf("Encryption Key Used: %i, Encyrption Key: %i\r\n", EncryptCnt,
Encryption[EncryptCnt]);
      Data[8] = Data[8]^Encryption[EncryptCnt];
      printf("Decrypted Header: %i \r\n", Data[8]);
      EncryptCnt++;
      EncryptCnt = EncryptCnt%32;

      printf("Encryption Key Used: %i, Encyrption Key: %i\r\n", EncryptCnt,
Encryption[EncryptCnt]);
      Data[9] = Data[9]^Encryption[EncryptCnt];
      printf("Decrypted CTRL1: %i \r\n", Data[9]);
      EncryptCnt++;
      EncryptCnt = EncryptCnt%32;

      printf("Encryption Key Used: %i, Encyrption Key: %i\r\n", EncryptCnt,
Encryption[EncryptCnt]);
      Data[10] = Data[10]^Encryption[EncryptCnt];
      printf("Decrypted CTRL2: %i \r\n", Data[10]);
      EncryptCnt++;
      EncryptCnt = EncryptCnt%32;

      printf("Encryption Key Used: %i, Encyrption Key: %i\r\n", EncryptCnt,
Encryption[EncryptCnt]);
      Data[11] = Data[11]^Encryption[EncryptCnt];
      printf("Decrypted CTRL3: %i \r\n", Data[11]);
      EncryptCnt++;
      EncryptCnt = EncryptCnt%32;

      StoreData();
}

void StoreEncr( void ){
      //Stores the data into the encryption array
      for(int i = 0; i<ENCR_LENGTH; i++){
            Encryption[i] = Data[i+RX_DATA_OFFSET+1];
      }
      EncryptCnt = 0;
}

void ResetEncr( void ){
      //resets index to 0 if synchronization is lost
      EncryptCnt = 0;
```

```
}


uint8_t getHeader( void ){
      return Header;
}

uint8_t getAPI( void ){
      return Frame_API;
}

uint8_t getSoftwareDogTag( void ){
      return RecDogTag;
}

uint8_t getLSBAddress( void ){
      return LSB_Address;
}

uint8_t getMSBAddress( void ){
      return MSB_Address;
}

uint8_t getPerData( void ){
      return PerData;
}

uint8_t getBrakeData( void ){
      return BrakeData;
}

uint8_t getMoveData( void ){
      return MoveData;
}

uint8_t getTurnData( void ){
      return TurnData;
}
```