```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * Motor PIC
 * Definitions
 * Language: C
 *
 * Blake English
 * Nina Joshi
 * Andi Kleissner
 */


#ifndef _DEFS_H_
#define _DEFS_H_

#include "bitdefs.h" // BIT0HI, BIT0LO....definitions
#include <htc.h>

/* Port A bits: */
#define SAFETY_CUTOFF_LED_BIT 0
#define SAFETY_CUTOFF_LED_DDR TRISC
#define SAFETY_CUTOFF_LED_PORT PORTC

/* Port B bits: */
#define ASYNC_TX_BIT 7
#define ASYNC_TX_DDR TRISB
#define ASYNC_TX_PORT PORTB

#define ASYNC_RX_BIT 5
#define ASYNC_RX_DDR TRISB
#define ASYNC_RX_PORT PORTB

/*Port C bits*/
#define PWM_OUTPUT_BIT 5
#define PWM_OUTPUT_DDR TRISC
#define PWM_OUTPUT_PORT PORTC

#define MOTOR_NON_PWM_OUTPUT_BIT 4
#define MOTOR_NON_PWM_OUTPUT_DDR TRISC
#define MOTOR_NON_PWM_OUTPUT_PORT PORTC


/* Convenience */
#define FALSE 0
#define TRUE 1

#define LOW 0
#define HIGH 1

#define CCW 0
#define CW 1

#define FWD 1
#define REV 2
```

```
#define RIGHT 3
#define LEFT 4

#define SETBIT(ADDRESS,BIT) (ADDRESS |= (1<<BIT)) //set bit (high)      //output
#define CLEARBIT(ADDRESS,BIT) (ADDRESS &= ~(1<<BIT)) //clear bit (low)  //input
#define CHECKBIT(ADDRESS,BIT) (ADDRESS & (1<<BIT)) //test single bit in I/O location
#define TOGGLEBIT(ADDRESS,BIT) (ADDRESS ^= (1<<BIT)) //toggle single bit

typedef  struct{
        unsigned char StarboardMotorDirection;
        unsigned char PortMotorDirection;

        unsigned char StarboardMotorSpeed;
        unsigned char PortMotorSpeed;
}motorSpeedValues;


/* Functions */
void initPortsAndPins(void);
void setupTimer1(void);
void initPWM(void);
void updateTheMotorPWMSettings(void);
unsigned short getBigTime(unsigned char command);
void MAIN_setNewMotorDriveValues(motorSpeedValues motorValues);
unsigned char MAIN_getTheStateOfMotorSafteyCutoffFlag(void);
void MAIN_assignTheStateOfMotorSafteyCutoffFlag(unsigned char flagState);
void MAIN_resetTheSafetyCounter();

#endif
```

```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * Motor PIC
 * Events Module
 * Language: C
 *
 * Blake English
 * Nina Joshi
 * Andi Kleissner
 */


#include <htc.h>
#include "defs.h"
#include "events.h"
#include "xbee.h"

static unsigned char crabsTransfered;
static unsigned char crabRate;

/* Functions */
/* Checks and returns any events that have occurred. */
Event_t EVENTS_CheckForEvents(void) {
	Event_t event = NO_EVENT;

	if(XBEE_HasAFullMessageBeenReceived() != 0) {
	    event = XBEE_FULL_MESSAGE_RECEIVED_EVENT;
	    return event;
	}

	if(XBEE_HasANewDriveCommandBeenReceived() != 0) {
	    return NEW_DRIVE_COMMAND_FROM_HC;
	}

	//if (event != NO_EVENT)
	    //return event;

	return event;
}

void EVENTS_RunStateMachine(Event_t event) {
	switch(event) {
	    case NO_EVENT:
			break;
	    case XBEE_FULL_MESSAGE_RECEIVED_EVENT:
			//we've gotten a message over the radio, now parse that data
			XBEE_InterperateReceivedMessage();
			break;
	    case NEW_DRIVE_COMMAND_FROM_HC:
			//update the motors
			MAIN_setNewMotorDriveValues(XBEE_getMostRecentMotorSpeedValues());
			MAIN_resetTheSafetyCounter(); //reset the counter
			if(MAIN_getTheStateOfMotorSafteyCutoffFlag()!= 0) { //if we are coming from a safety
cutoff state
```

```
                        MAIN_resetTheSafetyCounter(); //reset the counter
                        MAIN_assignTheStateOfMotorSafteyCutoffFlag(0);  //lower the flag
                        CLEARBIT(SAFETY_CUTOFF_LED_PORT,
SAFETY_CUTOFF_LED_BIT);  //turn off the LED
                }
                break;
        case SAFTEY_TIMER_EXPIRED_EVENT:
                //kill the motors
                break;
        default:
                break;
    }
}
```

```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * Motor PIC
 * Events Module
 * Language: C
 *
 * Blake English
 * Nina Joshi
 * Andi Kleissner
 */


#ifndef _EVENTS_H_
#define _EVENTS_H_

typedef enum {
	NO_EVENT,
	XBEE_FULL_MESSAGE_RECEIVED_EVENT,
	NEW_DRIVE_COMMAND_FROM_HC,
	SAFTEY_TIMER_EXPIRED_EVENT
} Event_t;

Event_t EVENTS_CheckForEvents(void);
void EVENTS_RunStateMachine(Event_t event);

#endif
```

```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * Motor PIC
 * Main Module
 * Language: C
 *
 * Blake English
 * Nina Joshi
 * Andi Kleissner
 */


#include <htc.h>
#include "bitdefs.h" // BIT0HI, BIT0LO....definitions
#include "defs.h"
#include "PirateAsync.h"
#include "xbee.h"
#include "events.h"

//CHANGE WHAT IS DEFINED BELOW TO CHANGE WHICH PICK IS BEING PROGRAMMED
//#define PIC_IS_CONTROLLING_STARBOARD_MOTOR
#define PIC_IS_CONTROLLING_PORT_MOTOR

#define TIME_MOTOR_SAFTEY_CUTTOFF 10  //based on timer 1 overflow (0.1 sec, so the cutoff
happens after 1 second if it = 10)

__CONFIG(HS & WDTDIS & BORDIS); //can only be called once

unsigned short timeSinceLastValidMessage = TIME_MOTOR_SAFTEY_CUTTOFF + 1;  //this is set high
on purpose so that we have to get a valid

                                        //message before we start going

                                        //there is still nothing to prevent the boat from starting at

                //wide open throttle if the HC is being abused...etc
unsigned char motorSafteyCutoffFlag = 0;
motorSpeedValues driveMotorValues;

void main (void) {
        initPortsAndPins();
   setupTimer1();
        initPWM();
        XBEE_Init();

        ei(); //enable global interrupts
        PEIE=1;  //all peripheral interrupts are enabled
        TMR1IE = 1; //enable timer 1 overflow interrupt (for long time counting using timer 1 overflows to
increment a counter)
        RCIE = 1;  //Enable RX interrupt

        //TURN INDICATOR LED OFF
        CLEARBIT(SAFETY_CUTOFF_LED_PORT, SAFETY_CUTOFF_LED_BIT);
```

```
        //this is actually an event...
        driveMotorValues.StarboardMotorDirection = FWD;
        driveMotorValues.PortMotorDirection =  FWD;
        driveMotorValues.StarboardMotorSpeed = 0;
        driveMotorValues.PortMotorSpeed = 0;

        MAIN_setNewMotorDriveValues(driveMotorValues);
        for(;;){  //MAIN LOOP
            EVENTS_RunStateMachine(EVENTS_CheckForEvents());
            //if we haven't recived a message in a while, stop the motors
            if(timeSinceLastValidMessage >= TIME_MOTOR_SAFTEY_CUTTOFF &&
MAIN_getTheStateOfMotorSafteyCutoffFlag() == 0) {
                    //this is actually an event...
                    driveMotorValues.StarboardMotorDirection = FWD;
                    driveMotorValues.PortMotorDirection =  FWD;
                    driveMotorValues.StarboardMotorSpeed = 0;
                    driveMotorValues.PortMotorSpeed = 0;
                    //set the motor speeds to 0
                    //assign the 0 motor speed values
                    MAIN_setNewMotorDriveValues(driveMotorValues);
                    //illuminate the LED
                    SETBIT(SAFETY_CUTOFF_LED_PORT, SAFETY_CUTOFF_LED_BIT);
                    //raise the flag
                    MAIN_assignTheStateOfMotorSafteyCutoffFlag(1);
            }
        } //end main loop
} //end main

void interrupt int_res (void) {
        if(RCIF != 0) { //we've had a receive interrupt ASYNC
            XBEE_ReadIncommingAsyncByte(RCREG);  //FIXED this function is too slow!!!, there is an
overrun error.  (bitfield mouseover on RCSTA)
            RCIF = 0; //clear the interrupt flag
        }

        if(TXIF != 0) { //we've had a TX interrupt meaning the output buffer is empty and we can transmit
now
            TXIE = 0; //disable this interrupt
            TXIF = 0;
        }

        if(TMR1IF != 0) {
            //this is where we could keep a longer timer, ie: increment it here
            timeSinceLastValidMessage++;
            TMR1IF = 0;
        }
}

void MAIN_setNewMotorDriveValues(motorSpeedValues motorValues) {
        driveMotorValues = motorValues;
        updateTheMotorPWMSettings();
}

void MAIN_resetTheSafetyCounter(void) {
        timeSinceLastValidMessage = 0;
}
```

```
unsigned char MAIN_getTheStateOfMotorSafteyCutoffFlag(void) {
	return motorSafteyCutoffFlag;
}

void MAIN_assignTheStateOfMotorSafteyCutoffFlag(unsigned char flagState) {
	motorSafteyCutoffFlag = flagState;
}

void updateTheMotorPWMSettings(void) {
	//called when we've gotten a new value from the HC,
	//so update the pwm
	#ifdef PIC_IS_CONTROLLING_STARBOARD_MOTOR
	//we are the starboard motor
	if(driveMotorValues.StarboardMotorDirection == FWD) {
	    CCPR1L = driveMotorValues.StarboardMotorSpeed;
	    CLEARBIT(MOTOR_NON_PWM_OUTPUT_PORT, MOTOR_NON_PWM_OUTPUT_BIT);
	} else {
	    CCPR1L = (255-driveMotorValues.StarboardMotorSpeed);
	    SETBIT(MOTOR_NON_PWM_OUTPUT_PORT, MOTOR_NON_PWM_OUTPUT_BIT);
	}
	#endif


	#ifdef PIC_IS_CONTROLLING_PORT_MOTOR
	//we are the port motor
	if(driveMotorValues.PortMotorDirection == REV) {
	    CCPR1L = driveMotorValues.PortMotorSpeed;
	    CLEARBIT(MOTOR_NON_PWM_OUTPUT_PORT, MOTOR_NON_PWM_OUTPUT_BIT);
	} else {
	    CCPR1L = (255-driveMotorValues.PortMotorSpeed);
	    SETBIT(MOTOR_NON_PWM_OUTPUT_PORT, MOTOR_NON_PWM_OUTPUT_BIT);
	}
	#endif
}

void initPortsAndPins(void){
	PORTC = 0x00; //clear all of port c

	SETBIT(ASYNC_RX_DDR,  ASYNC_RX_BIT); //input
	CLEARBIT(ASYNC_TX_DDR, ASYNC_TX_BIT); //output

	CLEARBIT(PWM_OUTPUT_DDR, PWM_OUTPUT_BIT); //output
	CLEARBIT(MOTOR_NON_PWM_OUTPUT_DDR, MOTOR_NON_PWM_OUTPUT_BIT);
//output
	CLEARBIT(SAFETY_CUTOFF_LED_DDR, SAFETY_CUTOFF_LED_BIT); //output

	/*ANALOG INPUT PINS*/
	ANSEL = 0;  //NO ADC, all digital pins
	ANSELH = 0;  //NO ADC, all digital pins
}

void setupTimer1(void) {
	//Using timer 1 for the step control
	//Turn on timer 1
	//use internal instruction clock 20mhz/4 = 5mhz
```

```
	//use a /8 prescaler = .625Mhz
	di();
	T1CON = 0b10110101;		//setup the inital timer control bits
	TMR1H = 0x00; //clear timer high byte
	TMR1L = 0x00; //clear timer low byte
	PIR1 = 0x00; //clear interrupt flags
	ei();
}

void initPWM(void){
	//SINGLE channel PWM, one PIC per motor
	//1.Disable the PWM pin (CCP1) output driver by
	//setting the associated TRIS bit.
	/*PWM OUTPUT PIN*/
	SETBIT(PWM_OUTPUT_DDR, PWM_OUTPUT_BIT); //DISABLE OUTPUT DRIVER
	//2.Set the PWM period by loading the PR2 register.
	PR2 = 0xFF;  //full period
	//3.Configure the CCP module for the PWM mode
	//by loading the CCP1CON register with the appropriate values.
	//CCP1CON: ENHANCED CCP1 CONTROL REGISTER
	P1M1 = 0;
	P1M0 = 0;  //Single PWM output, 00 =Single output; P1A modulated; P1B, P1C, P1D assigned as
port pins
	CCP1M3 = 1;
	CCP1M2 = 1;  //PWM mode; P1A, P1C active-high; P1B, P1D active-high
	CCP1M1 = 0;
	CCP1M0 = 0;
	//4.Set the PWM duty cycle by loading the CCPR1L
	//register and DC1B<1:0> bits of the CCP1CON register.
	CCPR1L = 100;
	DC1B1 = 0;
	DC1B0 = 0;
	//5.Configure and start Timer2:
	//ïClear the TMR2IF interrupt flag bit of the PIR1 register.
	TMR2IF = 0;
	//ïSet the Timer2 prescale value by loading the
	//T2CKPS bits of the T2CON register.
	//SET PRESCALE to 1 for about 20KHz pwm
	T2CKPS1 = 0;
	T2CKPS0 = 0; //prescaler of 1
	//ïEnable Timer2 by setting the TMR2ON bit of the T2CON register.
	TMR2ON = 1;
	//6.Enable PWM output after a new PWM cycle has started:
	//ïWait until Timer2 overflows (TMR2IF bit of the PIR1 register is set).
	//ïEnable the CCP1 pin output driver by clearing the associated TRIS bit.
	//DON't wait for the overflow, we can accept a first pulse that is messed up a little...
	/*PWM OUTPUT PIN*/
	CLEARBIT(PWM_OUTPUT_DDR, PWM_OUTPUT_BIT); //ENABLE output driver
}
```

```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * Motor PIC
 * Async Module
 * Language: C
 *
 * Blake English
 * Nina Joshi
 * Andi Kleissner
 */


#include "PirateAsync.h"

usartTxMessage TxUSARTmessageBuffer;
static unsigned char sendingUSARTmessageFlag = 0;

//Higher level USART transmitter
//Take in a data structure
//send that data structure
//indicate when sending is done
//disable TX interrupts when not sending
//enable TX interrupt when we want to send something
//send one byte at a time until we are done, then disable TX interrupts

void ASYNC_sendUSARTmessage(usartTxMessage message) {
	sendingUSARTmessageFlag = 1; //raise the flag that indicates that we are sending a message
	//transfer USART message to TX message buffer
	TxUSARTmessageBuffer = message;
	TxUSARTmessageBuffer.messageIndex = 0;  //reset the index
	//enable TX interrupts
	TXIE = 1;
}

void ASYNC_finishedSendingUSARTmessage(void) { //do the cleanup assocated with finishing sending a
message
	//disable TX interrupts
	TXIE = 0;
	sendingUSARTmessageFlag = 0; //raise the flag that indicates that we are sending a message
}

unsigned char ASYNC_areWeSendingAUSARTmessage(void) {
	return sendingUSARTmessageFlag;
}

void ASYNC_sendNextUSARTTxMessageByte(void) {
	//this will send the data byte in the message buffer that is being pointed to with the index value
	ASYNC_sendUSARTbyte(TxUSARTmessageBuffer.messageByte[TxUSARTmessageBuffer.messa
geIndex]);
	//increase the index count
	if(TxUSARTmessageBuffer.messageIndex >= (TxUSARTmessageBuffer.numberOfBytes-1)) { //we
are at the last byte of the message
		//we have finished sending the message, so cleanup
		ASYNC_finishedSendingUSARTmessage();
```

```
        } else {
            TxUSARTmessageBuffer.messageIndex++;
        }
}

void ASYNC_sendUSARTbyte(unsigned char dataByteToSend) {
        //do some sort of MAC or line checking here...if we need to
        TXREG = dataByteToSend;
}

void ASYNC_setupUSART(void) {
        di();
            /*Baud rate setup*/
            SCKP = 0; //transmit non-inverted data
            ABDEN = 0; //autobaud functionality turned off
            WUE = 0; //normal receiver operation
        //  BRG16 = 0; //8 bit baud generator used
        //  SPBRG = 63; //Set USART speed  9.6 kbps, .16% error, FOSC 20MHz


        //  BRG16 = 1;
        //  BRGH = 0;
        //  SPBRG = 129; //Set USART speed  9.6 kbps, .16% error, FOSC 20MHz, 5V

            BRG16 = 0;
            BRGH = 1;
            SPBRG = 129; //Set USART speed  9.6 kbps, .16% error, FOSC 20MHz, 5V


            /*TRANSMITTER*/
            //TXSTA REGISTER
                    TX9 = 0; //8 bit transmission
                    SYNC = 0; //async mode
                    //BRGH = 1; //High Speed moded
                    TRMT = 1; //Transmit shift reg
                    TXEN = 1; //enable transmission
                    TXIF = 0; //lower the transmit flag
            /*RECIVER*/
            //RCSTA REGISTER
                    RX9 = 0; //8 bit reception
                    ADDEN = 0; //disable address detection wakup
                    FERR = 0; //framing error flag lowered
                    OERR = 0; //overrun error flag lowered
                    SPEN = 1;           //enable serial port pins
                    CREN = 1; //Enable continious reception
                    RCIF = 0; //clear the flag

            /*INTERRUPTS*/
                    RCIE = 1;  //Enable RX interrupt
        ei();
}
```

```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * Motor PIC
 * Async Module
 * Language: C
 *
 * Blake English
 * Nina Joshi
 * Andi Kleissner
 */


//This function requres timer 1 to be setup and running...it is used for delays and such
//this can be changed later to timer 2 or 0 or ....etc, but not right now

#ifndef _PIRATE_ASYNC_H_
#define _PIRATE_ASYNC_H_

#include <htc.h>
#include "defs.h"  //this is only requred if the pins are not defined
                                        //in PirateLCD.h

typedef struct{
  unsigned char messageByte[20];
  unsigned char messageIndex;
  unsigned char numberOfBytes;
}usartTxMessage;

/*functions*/
void ASYNC_setupUSART(void);
void ASYNC_sendUSARTmessage(usartTxMessage message);
void ASYNC_finishedSendingUSARTmessage(void);  //do the cleanup assocated with finishing sending a
message
unsigned char ASYNC_areWeSendingAUSARTmessage(void);
void ASYNC_sendNextUSARTTxMessageByte(void);
void ASYNC_sendUSARTbyte(unsigned char dataByteToSend);

#endif
```

```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * Motor PIC
 * Xbee Module
 * Language: C
 *
 * Blake English
 * Nina Joshi
 * Andi Kleissner
 */


#include "xbee.h"
#include "PirateAsync.h"

/* XBEE module transmits and receives communications via the XBEE radio transmitters.
 * Takes in a data structure, sends the structure one byte at a time. */

/* Variables */
static unsigned char rxMessageBuffer[RX_BUFFER_SIZE];  //or boat message length...I don't know which
one is bigger
static unsigned char rxMessageBufferIndex = 0;
static unsigned char rxCompleteFlag = 0;
static unsigned char rxLengthOfIncommingMessage = 0;
static unsigned char rxLengthOfIncommingMessageFrame = 0;
static unsigned char MessageBeingRecivedFlag = NO_MESSAGE;
static unsigned char FullMessageRecivedFlag = NO_MESSAGE;
static unsigned char newDriveCommandFlag = 0;
motorSpeedValues lastReceivedMotorSpeedValues;

/* Functions */
/* Initialize XBEE communications */
void XBEE_Init(void) {
        ASYNC_setupUSART();
        TXIE = 0;  //disable TX interrupt
}

/* Checks to see if a message is in the process of being transferred. */
unsigned char XBEE_IsOutgoingRadioMessageInProgress(void) {
        return ASYNC_areWeSendingAUSARTmessage();
}

/* Checks to see if a message is in the process of being transferred. */
unsigned char XBEE_HasAFullMessageBeenReceived(void) {
        if(FullMessageRecivedFlag == MESSAGE_RECIVED) {
            //lower the flag if the message has been checked
            FullMessageRecivedFlag = NO_MESSAGE;
            return MESSAGE_RECIVED;
        } else {
            return NO_MESSAGE;
        }
}

unsigned char XBEE_HasANewDriveCommandBeenReceived(void) {
```

```
        if(newDriveCommandFlag != 0) {
            //lower the flag if the message has been checked
            newDriveCommandFlag = 0;
            return 1;
        } else {
            return 0;
        }
}

motorSpeedValues XBEE_getMostRecentMotorSpeedValues(void) {
        return lastReceivedMotorSpeedValues;
}

/* Transfers the next byte in the message. When end of message has been reached, will stop sending. */
void XBEE_SendNextMessageByte(void) {
        //this will send the data byte in the message buffer that is being pointed to with the index value
        ASYNC_sendNextUSARTTxMessageByte();
}

/* RX */
/* Reads in message byte by byte on the receive register and stores them in the rxMessageBuffer array. */
void XBEE_ReadIncommingAsyncByte(unsigned char rxByte) {
        //rxByte = RCREG;  //read in the data
        if(MessageBeingRecivedFlag == NO_MESSAGE) {
            //this is the start of a new message
            //check to see if the start delimiter is appoprate
            if(rxByte == RADIO_MESSAGE_START_DELIMETER) {
                        //this is a radio async message, and is also the first byte.  what luck!
                        //store the data
                        rxMessageBuffer[0] = RADIO_MESSAGE_START_DELIMETER;
                        //reset the index (we are setting it to 1 because we just stored something in 0)
                        rxMessageBufferIndex = 1;
                        //raise the message being recived flag
                        MessageBeingRecivedFlag = MESSAGE_BEING_RECIVED;
                        rxLengthOfIncommingMessage = 12;  //this will get us past the index of the message
length LSB which will update the message length (hopefully)
                        rxLengthOfIncommingMessageFrame = 12;
            } else {
                        //either we've messed something up, we got bad data, or something else non-radio is
talking on the line
                        //error flag
                        //don't reset index, store data, or raise flags
            }
        } else {
            //this is the continuation byte of a message (because the message being recived flag is raised)
            //which byte is this?  Length?  Type?  Broadcast/Not?
            //store the message byte
            rxMessageBuffer[rxMessageBufferIndex] = rxByte;  //store the next byte into the incomming
message buffer
            if(rxMessageBufferIndex == INDEX_INCOMMING_MSG_LENGTH_LSB) {
                        //I think that just the LSB will be enough to read in the message length
                        rxLengthOfIncommingMessageFrame = rxMessageBuffer[rxMessageBufferIndex];
                        rxLengthOfIncommingMessage = rxLengthOfIncommingMessageFrame + 4;
            }
            if(rxMessageBufferIndex == rxLengthOfIncommingMessage - 1) { //we have reached the end of
the message (including the checksum)
```

```
                        //reset
                        //raise and lower flags
                        MessageBeingRecivedFlag = NO_MESSAGE;
                        FullMessageRecivedFlag = MESSAGE_RECIVED;
                        //call something here to parse the data
                        //XBEE_InterperateRecivedMessage();
                }else{
                        //we are still in the incomming message
                        //increment the message index so that the next byte is stored correctly
                         rxMessageBufferIndex++;
                        if(rxMessageBufferIndex >= (RX_BUFFER_SIZE)) { //if the index has reached the end
of the array size (the buffer)
                                 rxMessageBufferIndex = RX_BUFFER_SIZE - 1;  //prevent us from exceeding
the size of the array
                                //there has been some sort of error if we got here,
                                MessageBeingRecivedFlag = NO_MESSAGE;  //reset things
                         }
                }
        }
}

void XBEE_InterperateReceivedMessage(void) {
        //this function will decide what type of message was recived, and from whom, and extract the
imporant data from it
        unsigned char msgLength;
        unsigned char msgFrameLength;
        unsigned char bufferContents[RX_BUFFER_SIZE];
        unsigned char j;
        unsigned char checkSum = 0;

        //we have to store the important data quickly so it doesn't get over-written by a new incomming
message, this would be a lot easier
        //if there were structures implemented
        msgLength = rxLengthOfIncommingMessage;
        msgFrameLength = rxLengthOfIncommingMessageFrame;

        for(j=0; j<msgLength; j++) {
            //transfer data over
            bufferContents[j] = rxMessageBuffer[j];
            if(j>=INDEX_INCOMMING_API_IDENT) {
                        //this adds the contents of the message frame so that we can check the checksum
                        checkSum += bufferContents[j];
            }
        }

        if(bufferContents[INDEX_INCOMMING_API_IDENT] ==
RADIO_MESSAGE_RECEIVE_API_IDENTIFIER) { //is the message API identifier correct for an
incomming radio message?
            if(checkSum != 0xFF) {
                        //the checksum was wrong...error here, don't pass on the data
                        //let someone know that the checksum was wrong
            } else {
                        //the checksum was right, finish the calculations and pass on the data to the approprate
people
                        //events ultimately, i'll set flags for now
                        //first, what type of message is it?
```

```
                //look at source address
                if(bufferContents[INDEX_INCOMMING_SOURCE_ADDR_MSB] ==
HC_RADIO_ADDRESS_MSB && bufferContents[INDEX_INCOMMING_SOURCE_ADDR_LSB] ==
HC_RADIO_ADDRESS_LSB) {
                        //if the source address of the radio message matches our HC's address
                        //is the message private?
                        //or broadcast?
                        //it doesn't really matter, if it is from our HC, it probably is private and it
probably
                        //is a control command
                        //which, the motor controller pics should be responding to
                        if(bufferContents[INDEX_INCOMMING_START_OF_DATA] ==
HCDCH_DRIVE_CMD_MSG_INDICTOR) {
                                //we have received a motor drive command message from the HC
                                //get the data
                                //lastReportedCrabOnBoard =
bufferContents[INDEX_INCOMMING_START_OF_DATA+1] +
bufferContents[INDEX_INCOMMING_START_OF_DATA+2];
                                //lastReportedWaterLoggedStatus =
bufferContents[INDEX_INCOMMING_START_OF_DATA+3];
                                //raise the flag
                                lastReceivedMotorSpeedValues.PortMotorDirection =
bufferContents[INDEX_INCOMMING_START_OF_DATA+1];
                                lastReceivedMotorSpeedValues.PortMotorSpeed =
bufferContents[INDEX_INCOMMING_START_OF_DATA+2];
                                lastReceivedMotorSpeedValues.StarboardMotorDirection =
bufferContents[INDEX_INCOMMING_START_OF_DATA+3];
                                lastReceivedMotorSpeedValues.StarboardMotorSpeed =
bufferContents[INDEX_INCOMMING_START_OF_DATA+4];
                                newDriveCommandFlag = 1;
                        }
                } else {
                        //it really doesn't matter what else is received, ignore it
                }
        } //end checksum check else
    } //end is the message API identifier correct for an incomming radio message?
}
```

```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * Motor PIC
 * Xbee Module
 * Language: C
 *
 * Blake English
 * Nina Joshi
 * Andi Kleissner
 */


#ifndef _XBEE_H_
#define _XBEE_H_

#include <htc.h>
#include "defs.h"
#include "PirateAsync.h"

//Messages between the HC and DCH or vicaversa
#define DCHHC_STATUS_MESSAGE_INDICTOR 0xF2
#define RX_BUFFER_SIZE 20

//flags
#define NO_MESSAGE 0
#define MESSAGE_RECIVED 1
#define MESSAGE_BEING_RECIVED 2

#define RADIO_MESSAGE_START_DELIMETER 0x7E

//Radio message API identifiers
#define RADIO_MESSAGE_SEND_API_IDENTIFIER 0x01
#define RADIO_MESSAGE_SEND_STATUS_API_IDENTIFIER 0x89
#define RADIO_MESSAGE_RECEIVE_API_IDENTIFIER 0x81

//incoming radio message defines
#define INDEX_INCOMMING_START_DELIMETER 0
#define INDEX_INCOMMING_MSG_LENGTH_MSB 1
#define INDEX_INCOMMING_MSG_LENGTH_LSB 2
#define INDEX_INCOMMING_API_IDENT 3
#define INDEX_INCOMMING_SOURCE_ADDR_MSB 4
#define INDEX_INCOMMING_SOURCE_ADDR_LSB 5
#define INDEX_INCOMMING_SIGNAL_STRENGTH 6
#define INDEX_INCOMMING_OPTIONS_BYTE 7
#define INDEX_INCOMMING_START_OF_DATA 8

//Messages between the HC and DCH or vicaversa
#define HCDCH_DRIVE_CMD_MSG_INDICTOR 0xF1
#define DCHHC_STATUS_MESSAGE_INDICTOR 0xF2
#define DCHHC_SECURITY_KEY_MESSAGE 0xF4
#define DCHHC_PRIVATE_CRAB_MESSAGE 0xF3

/*RADIO ADDRESSES*/
#define BROADCAST_RADIO_ADDRESS 0xFF
```

```
#define HC_RADIO_ADDRESS_MSB 0x20
#define HC_RADIO_ADDRESS_LSB 0x86

#define DCH_RADIO_ADDRESS_MSB 0x21
#define DCH_RADIO_ADDRESS_LSB 0x86

#define NEPTUNE_RADIO_ADDR_MSB 0x77
#define NEPTUNE_RADIO_ADDR_LSB 0x77


/* Functions */
void XBEE_Init(void);
usartTxMessage XBEE_populateRadioMessage(unsigned char MsgAddrHigh, unsigned char
MsgAddrLow, unsigned char MsgData[], unsigned char MsgDataLength);
void XBEE_ProccessPrivateMessageFromNeptune(unsigned char data1, unsigned char complement1,
unsigned char data2, unsigned char complement2);
unsigned char XBEE_IsOutgoingRadioMessageInProgress(void);
unsigned char XBEE_HasAFullMessageBeenReceived(void);
void XBEE_SendNextMessageByte(void);
void XBEE_ReadIncommingAsyncByte(unsigned char rxByte);
void XBEE_InterperateReceivedMessage(void);
unsigned char XBEE_HasANewDriveCommandBeenReceived(void);
motorSpeedValues XBEE_getMostRecentMotorSpeedValues(void);

#endif
```