```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * HC 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 STEERTING_ANALOG_INPUT_BIT 0
#define STEERTING_ANALOG_INPUT_DDR TRISA
#define STEERTING_ANALOG_INPUT_PORT PORTA

#define THROTTLE_ANALOG_INPUT_BIT 2
#define THROTTLE_ANALOG_INPUT_DDR TRISA
#define THROTTLE_ANALOG_INPUT_PORT PORTA

/* 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 LCD_DB4_BIT 0
#define LCD_DB4_DDR TRISC
#define LCD_DB4_PORT PORTC

#define LCD_DB5_BIT 1
#define LCD_DB5_DDR TRISC
#define LCD_DB5_PORT PORTC

#define LCD_DB6_BIT 2
#define LCD_DB6_DDR TRISC
#define LCD_DB6_PORT PORTC

#define LCD_DB7_BIT 3
#define LCD_DB7_DDR TRISC
#define LCD_DB7_PORT PORTC

#define LCD_RS_BIT 4
```

```
#define LCD_RS_DDR TRISC
#define LCD_RS_PORT PORTC

#define LCD_Enable_BIT 5
#define LCD_Enable_DDR TRISC
#define LCD_Enable_PORT PORTC

#define HAT_LIMIT_SWITCH_BIT 6
#define HAT_LIMIT_SWITCH_DDR TRISC
#define HAT_LIMIT_SWITCH_PORT PORTC


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

#define LOW 0
#define HIGH 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

//placements for LCD stuff
#define LCD_START_OF_NUMBER_OF_CRAB_COL 6
#define LCD_START_OF_NUMBER_OF_CRAB_ROW 0

#define LCD_START_OF_STATUS_MSG_COL 11
#define LCD_START_OF_STATUS_MSG_ROW 0

#define IMPT_EVNT_NO_EVENT 0x00
#define IMPT_EVNT_WHITE_TEAM_ASSIGNED 0x01
#define IMPT_EVNT_RED_LEADER 0x02
#define IMPT_EVNT_RED_NOT_LEADER 0x03
#define IMPT_EVNT_GREEN_LEADER 0x04
#define IMPT_EVNT_GREEN_NOT_LEADER 0x05
#define IMPT_EVNT_DRY_DOCK_CARD_FIXED 0x06
#define IMPT_EVNT_UNLOADING_DOCK_UNLOADED 0x07
#define IMPT_EVNT_CRAB_POT_HAULLED 0x08
#define IMPT_EVNT_DCH_WATERLOGGED 0x09
#define IMPT_EVNT_DCH_UNWATERLOGGED 0x0A

#define CRAB_POT_0 0xA0
#define CRAB_POT_1 0xA1
#define CRAB_POT_2 0xA2
#define CRAB_POT_3 0xA3
#define CRAB_POT_4 0xA4
#define CRAB_POT_5 0xA5
#define CRAB_POT_6 0xA6
#define CRAB_POT_7 0xA7
```

```
#define CRAB_POT_8 0xA8
#define CRAB_POT_9 0xA9
#define CRAB_POT_10 0xAA
#define CRAB_POT_11 0xAB
#define CRAB_POT_12 0xAC
#define LOADING_DOCK 0xAD
#define DRY_DOCK 0xAE
#define RED_TEAM_CARD 0x22
#define GREEN_TEAM_CARD 0x44
#define WHITE_TEAM_CARD 0x11
#define NO_MATCH 0x55    //ANDI ADDED THIS TO THE BIGGER LIST

#define RED_FLEET 0x11
#define GREEN_FLEET  0x22
#define WHITE_FLEET 0x44

typedef  struct{
        unsigned char StarboardMotorDirection;
        unsigned char PortMotorDirection;

        unsigned char StarboardMotorSpeed;
        unsigned char PortMotorSpeed;
}motorSpeedValues;

void MAIN_setBoatCrabCount(unsigned int unitsOfCrab);
void MAIN_setBoatCrabCountToZero(void);
void MAIN_respondToNewCrabDataEvent(void);
unsigned int MAIN_getNumberOfCrabsOnBoat(void);
void MAIN_setWaterLoggedFlag(void);
void MAIN_clearWaterLoggedFlag(void);
void MAIN_setTheSateOfTheWaterLoggedFlag(unsigned char flagState);
void setupADC(void);
motorSpeedValues mixThrottleAndSteeringForMotors(unsigned char Throttle, unsigned char Steering);
unsigned char scaleADCValue(unsigned char ADCValue, unsigned char centerValue, unsigned char
deadZone, unsigned char lowerLimit, unsigned char upperLimit);
void respondToADCgoDone(void);
void MAIN_debugDriveMotorOutputsOnLCD(void);
unsigned char MAIN_getWaterLoggedFlagStatus(void);
void MAIN_AssignFleetAffiliation(unsigned char fleet);
void  MAIN_AssignXORCypher(unsigned char cypher);
unsigned char MAIN_GetFleetAffiliation(void);
unsigned char MAIN_GetXORCypher(void);
unsigned char MAIN_EncodeOrDecodeByteUsingCyper(unsigned char dataByte);
unsigned char MAIN_GetIsHatOnHCFlag(void);
unsigned char MAIN_GetLastHatSwitchState(void);
void MAIN_AssignLastHatSwitchState(unsigned char flagState);
void MAIN_AssignIsHatOnHCFlag(unsigned char flagState);
void MAIN_initalizeLCDTextForGamePlay(void);
void MAIN_initalizeLCDTextForHatOnHC(void);

#endif
```

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


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

static unsigned char crabsTransfered;
static unsigned char crabRate;
static unsigned char lastWaterLoggedStatus = 0;

/* 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_CheckNeptuneOpenSeasonFlag() != 0) {
	    event = NEPTUNE_OPEN_SEASON_BROADCAST_EVENT;
	    return event;
	}

	if(XBEE_HasNewDCHStatusMessageBeenReceived() != 0) {
	    LCD_setCursorPosition(1, 0);
	    LCD_writeAsciToLCD('S');   //if we ever see this it's bad
	    event = NEW_STATUS_MESSAGE_FROM_DCH_EVENT;
	    return event;
	}

	if(MAIN_getWaterLoggedFlagStatus() != 0 && lastWaterLoggedStatus == 0) {
	    //we've had a waterlogged event
	    lastWaterLoggedStatus = 1;
	    event = DCH_WATERLOGGED_EVENT;
	    return event;
	}

	if(MAIN_getWaterLoggedFlagStatus() == 0 && lastWaterLoggedStatus != 0) {
	    //we've had a un-waterlogged event
```

```
            lastWaterLoggedStatus = 0;
            event = DCH_UN_WATERLOGGED_EVENT;
            return event;
        }

        if(XBEE_HaveWeHadANewCrabPotFleetUpdate() != 0) {
            event = NEW_FLEET_CRAB_REPORT;
            return event;
        }

        //This is hat switch stuff, being used mostly for debugging now
        if(CHECKBIT(HAT_LIMIT_SWITCH_PORT, HAT_LIMIT_SWITCH_BIT) == 0 &&
MAIN_GetLastHatSwitchState() == 1) {
            MAIN_AssignLastHatSwitchState(0);
            event = HC_HAT_PLACED_ON_HC_EVENT;
            return event;
        }

        //This is still hat switch stuff, being used mostly for debugging now
        if(CHECKBIT(HAT_LIMIT_SWITCH_PORT, HAT_LIMIT_SWITCH_BIT) != 0 &&
MAIN_GetLastHatSwitchState() == 0) {
                    MAIN_AssignLastHatSwitchState(1);
                    event = HC_HAT_REMOVED_EVENT;
                    return event;
        }

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

        return event;
}

/* Events debug. */
void EVENTS_Debug(void){
        for(;;) {
            switch(EVENTS_CheckForEvents()) {
                    case NO_EVENT:
                            break;
                    default:
                            break;
            }
        }
}

void EVENTS_RunHatIsOnStateMachine(Event_t event) {
        switch(event) {
            case NO_EVENT:
            break;
        case HC_HAT_PLACED_ON_HC_EVENT:
            MAIN_initalizeLCDTextForHatOnHC();
            MAIN_AssignIsHatOnHCFlag(1);
            break;
        case HC_HAT_REMOVED_EVENT:
            MAIN_initalizeLCDTextForGamePlay();
            MAIN_AssignIsHatOnHCFlag(0);
            break;
```

```
        default:
            break;
        }
}

void EVENTS_RunHatIsOffStateMachine(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 NEPTUNE_OPEN_SEASON_BROADCAST_EVENT:
                        //open season!  go go go go go go go!
                        LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                        LCD_writeAsciToLCD('O');
                        LCD_writeAsciToLCD('P');
                        LCD_writeAsciToLCD('E');
                        LCD_writeAsciToLCD('N');
                        LCD_writeAsciToLCD(' ');
                        LCD_writeAsciToLCD('S');
                        LCD_writeAsciToLCD('S');
                        LCD_writeAsciToLCD('N');
                        LCD_writeAsciToLCD(' ');
                        break;
            case HC_HAT_PLACED_ON_HC_EVENT:
                        MAIN_initalizeLCDTextForHatOnHC();
                        MAIN_AssignIsHatOnHCFlag(1);
                        break;
            case HC_HAT_REMOVED_EVENT:
                        MAIN_initalizeLCDTextForGamePlay();
                        MAIN_AssignIsHatOnHCFlag(0);
                        break;
            case NEW_STATUS_MESSAGE_FROM_DCH_EVENT:
                        //update crab count and waterlogged flag
                        //update crab count
                        MAIN_setBoatCrabCount(XBEE_getLastCrabCountFromDCH());
                        //update the waterlogged flag state
                        //this should be more event driven ultimatly
                        //perhaps make an event when the flag changes state

        MAIN_setTheSateOfTheWaterLoggedFlag(XBEE_getLastWaterLoggedStatusFromDCH());
                        MAIN_AssignXORCypher(XBEE_getLastReportedXORCypher());
                        //set what fleet we are on
                        MAIN_AssignFleetAffiliation(XBEE_getLastReportedFleetAssignment());
                        LCD_setCursorPosition(1, 0);
                        LCD_setCursorPosition(LCD_START_OF_NUMBER_OF_CRAB_ROW,
LCD_START_OF_NUMBER_OF_CRAB_COL);
                        LCD_writeDecimalIntNumberToLCD(MAIN_getNumberOfCrabsOnBoat());
                        switch(XBEE_getLastReportedImptEvent()) {
                                    case IMPT_EVNT_NO_EVENT:
                                                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                                                LCD_writeAsciToLCD('N');
```

```
                LCD_writeAsciToLCD('O');
                LCD_writeAsciToLCD(' ');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('V');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('N');
                LCD_writeAsciToLCD('T');
                LCD_writeAsciToLCD(' ');
                break;
            case IMPT_EVNT_WHITE_TEAM_ASSIGNED:
                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                LCD_writeAsciToLCD('W');
                LCD_writeAsciToLCD('H');
                LCD_writeAsciToLCD('T');
                LCD_writeAsciToLCD(' ');
                LCD_writeAsciToLCD('T');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('A');
                LCD_writeAsciToLCD('M');
                LCD_writeAsciToLCD(' ');
                break;
            case IMPT_EVNT_RED_LEADER:
                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                LCD_writeAsciToLCD('R');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('D');
                LCD_writeAsciToLCD(' ');
                LCD_writeAsciToLCD('L');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('A');
                LCD_writeAsciToLCD('D');
                LCD_writeAsciToLCD(' ');
                break;
            case IMPT_EVNT_RED_NOT_LEADER:
                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                LCD_writeAsciToLCD('R');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('D');
                LCD_writeAsciToLCD(' ');
                LCD_writeAsciToLCD('F');
                LCD_writeAsciToLCD('L');
                LCD_writeAsciToLCD('W');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('R');
                break;
            case IMPT_EVNT_GREEN_LEADER:
                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                LCD_writeAsciToLCD('G');
                LCD_writeAsciToLCD('R');
                LCD_writeAsciToLCD('N');
                LCD_writeAsciToLCD(' ');
                LCD_writeAsciToLCD('L');
```

```
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('A');
                LCD_writeAsciToLCD('D');
                LCD_writeAsciToLCD(' ');
                break;
            case IMPT_EVNT_GREEN_NOT_LEADER:
                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                LCD_writeAsciToLCD('G');
                LCD_writeAsciToLCD('R');
                LCD_writeAsciToLCD('N');
                LCD_writeAsciToLCD(' ');
                LCD_writeAsciToLCD('F');
                LCD_writeAsciToLCD('L');
                LCD_writeAsciToLCD('W');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('R');
                break;
            case IMPT_EVNT_UNLOADING_DOCK_UNLOADED:
                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                LCD_writeAsciToLCD('U');
                LCD_writeAsciToLCD('N');
                LCD_writeAsciToLCD('L');
                LCD_writeAsciToLCD('O');
                LCD_writeAsciToLCD('A');
                LCD_writeAsciToLCD('D');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('D');
                LCD_writeAsciToLCD(' ');
                break;
            case IMPT_EVNT_CRAB_POT_HAULLED:
                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                LCD_writeAsciToLCD('P');
                LCD_writeAsciToLCD('O');
                LCD_writeAsciToLCD('T');
                LCD_writeAsciToLCD(' ');
                LCD_writeAsciToLCD('H');
                LCD_writeAsciToLCD('A');
                LCD_writeAsciToLCD('U');
                LCD_writeAsciToLCD('L');
                LCD_writeAsciToLCD('D');
                break;
            case IMPT_EVNT_DRY_DOCK_CARD_FIXED:
                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                LCD_writeAsciToLCD('R');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('P');
                LCD_writeAsciToLCD('A');
                LCD_writeAsciToLCD('R');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('D');
                LCD_writeAsciToLCD(' ');
                LCD_writeAsciToLCD(' ');
```

```
                        break;
                    case IMPT_EVNT_DCH_WATERLOGGED:
                        LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                        LCD_writeAsciToLCD('F');
                        LCD_writeAsciToLCD('L');
                        LCD_writeAsciToLCD('O');
                        LCD_writeAsciToLCD('O');
                        LCD_writeAsciToLCD('D');
                        LCD_writeAsciToLCD('E');
                        LCD_writeAsciToLCD('D');
                        LCD_writeAsciToLCD('!');
                        LCD_writeAsciToLCD(' ');
                        break;
                    default:
                        break;
                }
                break;
            case NEW_FLEET_CRAB_REPORT:
                LCD_setCursorPosition(1, 6);
                LCD_writeAsciToLCD('P');
                LCD_writeAsciToLCD('#');
                LCD_writeDecimalIntNumberToLCD(XBEE_getLastFleetReportedPotNumber());
                LCD_writeAsciToLCD(' ');
                LCD_writeAsciToLCD('R');
                LCD_writeAsciToLCD('T');
                LCD_writeDecimalIntNumberToLCD(XBEE_getLastFleetReportedRate());
                break;
            case DCH_WATERLOGGED_EVENT:
                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                LCD_writeAsciToLCD('F');
                LCD_writeAsciToLCD('L');
                LCD_writeAsciToLCD('O');
                LCD_writeAsciToLCD('O');
                LCD_writeAsciToLCD('D');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('D');
                LCD_writeAsciToLCD('!');
                LCD_writeAsciToLCD(' ');
                break;
            case DCH_UN_WATERLOGGED_EVENT:
                LCD_setCursorPosition(LCD_START_OF_STATUS_MSG_ROW,
LCD_START_OF_STATUS_MSG_COL);
                LCD_writeAsciToLCD('R');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('P');
                LCD_writeAsciToLCD('A');
                LCD_writeAsciToLCD('I');
                LCD_writeAsciToLCD('R');
                LCD_writeAsciToLCD('E');
                LCD_writeAsciToLCD('D');
                LCD_writeAsciToLCD(' ');
                break;
            default:
                break;
```

```
        }
}
```

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


#ifndef _EVENTS_H_
#define _EVENTS_H_

typedef enum {
	NO_EVENT,
	RFID_PIC_SIGNAL_LINE_HIGH_EVENT,
	RFID_PIC_SIGNAL_LINE_LOW_EVENT,
	NEPTUNE_OPEN_SEASON_BROADCAST_EVENT,
	NEPTUNE_DCH_UNLOADED_EVENT,
	NEPTUNE_DCH_REPAIRED_EVENT,
	NEPTUNE_CRAB_POT_DATA_RETURNED_EVENT,
	NEPTUNE_INVALID_KEY_EVENT,
	DCH_WATERLOGGED_EVENT,
	DCH_UN_WATERLOGGED_EVENT,
	WATERLOGGED_LINE_HI_FROM_SC,
	WATERLOGGED_LINE_LO_FROM_SC,
	HC_HAT_SWITCH_HIGH_EVENT,
	HC_HAT_SWITCH_LOW_EVENT,
	XBEE_FULL_MESSAGE_RECEIVED_EVENT,
	NEW_STATUS_MESSAGE_FROM_DCH_EVENT,
	NEW_PRIVATE_CRAB_POT_DATA_FROM_DHC_EVENT,
	NEW_DRIVE_COMMAND_FROM_HC,
	NEW_FLEET_CRAB_REPORT,
	HC_HAT_REMOVED_EVENT,
	HC_HAT_PLACED_ON_HC_EVENT
} Event_t;

Event_t EVENTS_CheckForEvents(void);
void EVENTS_Debug(void);
void EVENTS_RunHatIsOffStateMachine(Event_t event);
void EVENTS_RunHatIsOnStateMachine(Event_t event);
Event_t EVENTS_whatDidNeptuneRespondWith(void);

#endif
```

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


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

#define TRANS_TIME_HIGH_BYTE 0x90  //0x10

#define THROTTLE_ADC_CHANNEL 1
#define STEERING_ADC_CHANNEL 2
#define NO_ADC_CHANNEL_SELECTED 0

#define THROTTLE_ADC_CENTER 149
#define STEERING_ADC_CENTER 95
#define STEERING_ADC_DEAD_ZONE 1
#define THROTTLE_ADC_DEAD_ZONE 10
#define THROTTLE_ADC_UPPER_LIMIT 190 //201
#define THROTTLE_ADC_LOWER_LIMIT 0  //0
#define STEERING_ADC_UPPER_LIMIT 240
#define STEERING_ADC_LOWER_LIMIT 58

__CONFIG(HS & WDTDIS & BORDIS); //only call once

/* Variables */
static unsigned char lastHatSwitchState = 1;
static unsigned int numberOfCrabsOnBoat = 0;
static unsigned char boatIsWaterLoggedFlag = 0;
static unsigned char lastBoatEvent = 0;
unsigned char throttleADCValue = 0;
unsigned char steeringADCValue = 0;  //eventually change this so that it comes up at nominal midrage
values
unsigned char currentADCchannel = NO_ADC_CHANNEL_SELECTED;
unsigned char lastADCchannel = NO_ADC_CHANNEL_SELECTED;
unsigned char scaledThrottleValue = 0;
unsigned char scaledSteeringValue = 0;
motorSpeedValues driveMotorSpeeds;
unsigned char debugBlinkValue = 0;  //temp
unsigned char XORCypherKey = 0x00;
unsigned char FleetAffilation = WHITE_FLEET;
unsigned char isHatOnHCFlag = 0;
```

```
/* Function Prototypes */
void MAIN_Init(void);
void MAIN_SetupTimer1(void);
//void MAIN_Debug(void);

/* Functions */
void main (void) {
	di();
	MAIN_Init();
   MAIN_SetupTimer1();
	LCD_initLCDDisplay();
	XBEE_Init();
	setupADC();

	//Write out the permanent text to the screen
	//"CRAB:   0"
	MAIN_initalizeLCDTextForGamePlay();

	//set the current motor duty cycle and direction values to zero
	//this way they don't come up on full speed
	//they will get updated with the first ADC conversion and 5hz timer firing
	//after this
	driveMotorSpeeds.StarboardMotorDirection = FWD;
	driveMotorSpeeds.PortMotorDirection = FWD;
	driveMotorSpeeds.StarboardMotorSpeed = 0;
	driveMotorSpeeds.PortMotorSpeed = 0;

	//after all inits, enable interrupts
	ei(); //enable global interrupts
	PEIE = 1;  //all peripheral interrupts are enabled

   //Initialize the state machine and loop forever to run it.
	//STATE_InitStateMachine();
	//yes, we probably should.  couldn't find STATE, so I put the state machine in EVENTS

	for(;;) {
	   if (CHECKBIT(HAT_LIMIT_SWITCH_PORT, HAT_LIMIT_SWITCH_BIT) != 0) {
			EVENTS_RunHatIsOffStateMachine(EVENTS_CheckForEvents());
	   } else {
			EVENTS_RunHatIsOnStateMachine(EVENTS_CheckForEvents());
	   }

	   if(CCP1IF != 0) { //check to see if the compare flag is set?
			CCPR1H = TMR1H + TRANS_TIME_HIGH_BYTE;  //schedule next compare
			//debugBlinkValue causes a char on the LCD to alternate between '-' and '/'
			//and is a heartbeat to let us know we are still alive
			//its rate should be around 5HZ, use TRANS_TIME_HIGH_BYTE to modify it.
			if(debugBlinkValue == 0) {
				//update the LCD heartbeat
				LCD_setCursorPosition(1, 19);  //these seem to really slow it down...
				LCD_writeAsciToLCD('-'); //these seem to really slow it down...

				debugBlinkValue = 1;
				if(isHatOnHCFlag != 0) {
					//the hat is on the HC
					//set throttles to 0, 0
```

```
                                driveMotorSpeeds.StarboardMotorDirection = FWD;
                                driveMotorSpeeds.PortMotorDirection = FWD;
                                driveMotorSpeeds.StarboardMotorSpeed = 0;
                                driveMotorSpeeds.PortMotorSpeed = 0;
                        } else {
                                //the hat is off the HC
                                //calculate the new motor speeds based on the latest ADC scaled inputs
                                driveMotorSpeeds =
mixThrottleAndSteeringForMotors(scaledThrottleValue, scaledSteeringValue);
                        }
                        //if you need to see the motor DC and direction, use this:
                        //MAIN_debugDriveMotorOutputsOnLCD();  //adding this in messes with the
numbers, of course
                        //and then, send a drive command to the DCH

        XBEE_SendDriveCommandMessageToDCH(driveMotorSpeeds.PortMotorDirection,
driveMotorSpeeds.PortMotorSpeed, driveMotorSpeeds.StarboardMotorDirection,
driveMotorSpeeds.StarboardMotorSpeed);
                } else {
                        //update the LCD heartbeat
                        LCD_setCursorPosition(1, 19); //these seem to really slow it down...
                        LCD_writeAsciToLCD('/'); //these seem to really slow it down...
                        debugBlinkValue = 0;
                }
                //clear the timer flag
                //i'm not doing this in interrupts because it is not time critical and
                //the functions that are called are computationally intensive
                CCP1IF = 0;
        }
        //the ADC subsytem has finished a conversion, now we need to start another one
        if(GODONE == 0) { //we can start a conversion now
                //switch to the other ADC channel and start the next conversion
                respondToADCgoDone();
        }
    }//END OF MAIN FOR LOOP
}//END OF MAIN

//Here is the interrupt response routine
void interrupt int_res (void) {
    //Incomming async byte
    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
    }

    //outgoing async byte
    if(TXIF != 0) { //we've had a TX interrupt meaning the output buffer is empty and we can transmit
now
        //check to see if there actually is a message to send.
        //if(XBEE_IsOutgoingRadioMessageInProgress() == 0) {
        if(ASYNC_areWeSendingAUSARTmessage() == 0) {  //I had to change these to reduce the
number of function calls
                //we should never be getting here, but if we do, this will help to keep the system running
                TXIE = 0; //disable this interrupt
        } else {
```

```
                    //send the next byte if we are actually in a message
                    ASYNC_sendNextUSARTTxMessageByte();
                }
            }
}

void MAIN_Init(void){
        //clear all port registers so nothing comes up on
        PORTA = 0x00; //clear all of port a
        PORTB = 0x00; //clear all of port b
        PORTC = 0x00; //clear all of port c

        /*LCD PINS*/
        CLEARBIT(LCD_DB4_DDR, LCD_DB4_BIT); //output
        CLEARBIT(LCD_DB5_DDR, LCD_DB5_BIT); //output
        CLEARBIT(LCD_DB6_DDR, LCD_DB6_BIT); //output
        CLEARBIT(LCD_DB7_DDR, LCD_DB7_BIT); //output
        CLEARBIT(LCD_RS_DDR, LCD_RS_BIT); //output
        CLEARBIT(LCD_Enable_DDR, LCD_Enable_BIT); //output

        /*HAT SWITCH*/
        SETBIT(HAT_LIMIT_SWITCH_DDR, HAT_LIMIT_SWITCH_BIT); //input

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

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

        SETBIT(STEERTING_ANALOG_INPUT_DDR, STEERTING_ANALOG_INPUT_BIT); //input
        SETBIT(THROTTLE_ANALOG_INPUT_DDR, THROTTLE_ANALOG_INPUT_BIT); //input
        ANS0 = 1;  //Analog channel 0 selected for ADC  A0
        ANS1 = 1;   //analog channel 1 selected for ADC  A1
        ANS2 = 1;  //analog channel 2 selected for ADC A2
}

void MAIN_SetupTimer1(void) {
        //Using timer 1 for general purpose timing and delays
        //Turn on timer 1
        //use internal instruction clock 20mhz/4 = 5mhz
        //use a /8 prescaler = .625Mhz
        T1CON = 0b10110101;         //setup the inital timer control bits

        //now set it up for output compare, internally, software, no external pin
        CCP1CON = 0b00001010; //set the CCP1CON compare reg to output compare, software
        CCPR1H  = TRANS_TIME_HIGH_BYTE; //move the inital step time value into the upper byte of
the compare reg
        CCPR1L  = 0x00; //clear the lower byte of the compare time reg (we don't need the precision!)

        TMR1H = 0x00; //clear timer high byte
        TMR1L = 0x00; //clear timer low byte
        PIR1 = 0x00; //clear interrupt flags
}
```

```
//This sets the value of the crab count that the HC thinks the DCH has
void MAIN_setBoatCrabCount(unsigned int unitsOfCrab) {
        numberOfCrabsOnBoat = unitsOfCrab;
}

//this returns the number of crabs that the HC things the DCH has onboard
unsigned int MAIN_getNumberOfCrabsOnBoat(void) {
        return numberOfCrabsOnBoat;
}

//this sets/clears/assigns the state of the waterlogged flag (what the HC thinks the DCH is)
void MAIN_setWaterLoggedFlag(void) {
        boatIsWaterLoggedFlag = 1;
}
//this sets/clears/assigns the state of the waterlogged flag (what the HC thinks the DCH is)
void MAIN_clearWaterLoggedFlag(void) {
        boatIsWaterLoggedFlag = 0;
}

//this sets/clears/assigns the state of the waterlogged flag (what the HC thinks the DCH is)
void MAIN_setTheSateOfTheWaterLoggedFlag(unsigned char flagState) {
        boatIsWaterLoggedFlag = flagState;
}

//this returns the state of the waterlogged flag (what the HC thinks the DCH is)
unsigned char MAIN_getWaterLoggedFlagStatus(void) {
        return boatIsWaterLoggedFlag;
}

void setupADC(void) {
        //CONFIGURE THE ADC Module
        //select ADC conversion clock(ADCON1)
            ADCS2 = 1;
            ADCS1 = 1;  //use FOSC/64  (FOSC = 20mhz)
            ADCS0 = 0;

        //configure voltage reference(ADCON0 register)
            VCFG = 0;  //Vdd used as VREF
        //select ADC input channel(ADCON0 register)
            CHS3 = 0;
            CHS2 = 0;
            CHS1 = 0;
            CHS0 = 0;
        //select result format  (ADCON0 register)
            ADFM = 0;  //left justified result
            LCD_blockingDelay(100);  //5ms
        //turn on ADC.  Enable ADC
            ADON = 1;
            LCD_blockingDelay(100);  //5ms

        //CONFIG ADC INTERRUPT  (if we need to)
        //clear ADC interrupt flag
        //enable ADC interrupts
        //enable PIE
        //enable GIE
```

```
        //wait required aqusition time
        //GODONE = 1;  //set godone bit to start the conversion proccess
        //go/NOTdone = "GODONE"
}

void respondToADCgoDone(void) {
        switch(currentADCchannel) {
            case THROTTLE_ADC_CHANNEL:  //store the ADC result
                        throttleADCValue = ADRESH;
                        //update lcd screen
                        scaledThrottleValue = scaleADCValue(throttleADCValue,
THROTTLE_ADC_CENTER, THROTTLE_ADC_DEAD_ZONE, THROTTLE_ADC_LOWER_LIMIT,
THROTTLE_ADC_UPPER_LIMIT);
                        //setup for the next conversion
                        currentADCchannel = STEERING_ADC_CHANNEL;
                        //select ADC input channel(ADCON0 register)
                        CHS3 = 0;
                        CHS2 = 0;
                        CHS1 = 1;
                        CHS0 = 0;
                        VCFG = 1; //Vref = Vref Pin
                        break;
            case STEERING_ADC_CHANNEL:  //store the ADC result
                        steeringADCValue = ADRESH;
                        scaledSteeringValue = scaleADCValue(steeringADCValue,
STEERING_ADC_CENTER, STEERING_ADC_DEAD_ZONE, STEERING_ADC_LOWER_LIMIT,
STEERING_ADC_UPPER_LIMIT);
                        //setup pins for next conversion
                        currentADCchannel = THROTTLE_ADC_CHANNEL;
                        //select ADC input channel(ADCON0 register)
                        CHS3 = 0;
                        CHS2 = 0;
                        CHS1 = 0;
                        CHS0 = 0;
                        VCFG = 0; //Vref = Vdd (internal)
                        break;
            default:   //this happens the first time
                        //start the next conversion
                        //setup pins for next conversion
                        currentADCchannel = THROTTLE_ADC_CHANNEL;
                        //select ADC input channel(ADCON0 register)
                        CHS3 = 0;
                        CHS2 = 0;
                        CHS1 = 1;
                        CHS0 = 0;
                        VCFG = 1; //Vref = Vref Pin
                        break;
        }
        LCD_blockingDelay(10);  //5ms <--we can improve this
        //we could be updating something here instead of blocking delay...etc
        GODONE = 1;  //set godone bit to start the conversion proccess
}

//This function scales a value so that it takes up a full 0-255 char data range, this way we get the most
//out of the steering wheel and throttle ADC values even though the ADC values themselves don't go all the
```

```
way from
//0-255.  Right now it uses a float, but it could be made to use ints instead.  It takes about 10% of the
program
//and data memory
unsigned char scaleADCValue(unsigned char ADCValue, unsigned char centerValue, unsigned char
deadZone, unsigned char lowerLimit, unsigned char upperLimit) {
        unsigned char fullRange = 0;
        float scaler;

        unsigned int range;
        unsigned int value;
        unsigned char finalValue;

        if(ADCValue > upperLimit) {  //is the value bigger than the upper limit?
            return 255;  //return max of 255
        } else if(ADCValue < lowerLimit) {  //is the value smaller than the lower limit?
            return 0;    //return min of 255
        } else {
            //otherwise, it's outside of it's dead zone, so do the calculations
            //full range = Max - Min
            fullRange = upperLimit - lowerLimit;
            ADCValue = ADCValue - lowerLimit; //subtract lower limit to remove constant values
            //ints or ...would be much more space efficent than floats,
            //return range / value;   //would save 10% of data mem and 15% of pgm mem vs floats, do this
later (doesn't work now)
            scaler = (float) ADCValue / (float) fullRange;
            finalValue = (unsigned char) (scaler * 255);
            if(finalValue <= (127+deadZone) && finalValue >= (127 - deadZone)) {
                    //we are in the "deadZone"  watch out for zombies and other un-dead variables
                    finalValue = 127;
            }
            return finalValue;
        }
}

//this function mixes the scaled throttle value and the scaled steering value and outputs direction and
//dutycycle for the motors.  It converts fwd/back and right/left into differental motor values so that
//the boat can go forward, backwards, turn, or do any combination of them
motorSpeedValues mixThrottleAndSteeringForMotors(unsigned char Throttle, unsigned char Steering) {
        motorSpeedValues motor;
        unsigned char scaledThrottle;
        //unsigned char scaledSteering;
        signed int starboardMtrSpeed;
        signed int portMtrSpeed;
        unsigned char steeringDir;
        unsigned char throttleDir;
        unsigned char deltaSteering;

        //First, take the scaled inputs (0-255) from steering and throttle.
        //break them down into their component parts because 127 represents
        //neutral throttle or steering
        if(Throttle>=127) { //break down the throttle into positive and negitive (fwd/back)
            scaledThrottle = (Throttle - 127);
            throttleDir = FWD;
        } else {
            scaledThrottle = 127 - Throttle;
```

```
        throttleDir = REV;
    }

    if (Steering>=127) { //do the same for steering
        steeringDir = RIGHT;
        deltaSteering = (Steering - 127);
    } else {
        steeringDir = LEFT;
        deltaSteering = (127 - Steering);
    }

    //Add the throttle to the motor speeds (Fwd, or back)
    if(throttleDir == FWD) {
        portMtrSpeed = scaledThrottle;
        starboardMtrSpeed = scaledThrottle;
    } else { //rev
        portMtrSpeed = -scaledThrottle;
        starboardMtrSpeed = -scaledThrottle;
    }

    //now add the broken down steering components
    if(steeringDir == RIGHT) { //port motor faster than starboard motor
        portMtrSpeed += (deltaSteering);
        starboardMtrSpeed -= (deltaSteering); //throttle value - delta steering
    } else {
        portMtrSpeed -= (deltaSteering);
        starboardMtrSpeed += (deltaSteering); //throttle value - delta steering
    }

    //multiply by two to bring up from 127 max to 255 max
    starboardMtrSpeed = starboardMtrSpeed * 2;
    if(starboardMtrSpeed>=0) {
        //starboardMotor is going fwd
        if(starboardMtrSpeed>255) starboardMtrSpeed = 255;
        motor.StarboardMotorDirection = FWD;
        motor.StarboardMotorSpeed = (unsigned char) starboardMtrSpeed;
    } else {
        if(starboardMtrSpeed<-255) starboardMtrSpeed = -255;
        motor.StarboardMotorDirection = REV;
        motor.StarboardMotorSpeed = (unsigned char) (-starboardMtrSpeed);
    }

    portMtrSpeed = portMtrSpeed * 2;
    if(portMtrSpeed>=0) {
        if(portMtrSpeed>255) portMtrSpeed = 255;
        //starboardMotor is going fwd
        motor.PortMotorDirection = FWD;
        motor.PortMotorSpeed = (unsigned char) portMtrSpeed;
    } else {
        if(portMtrSpeed<-255) portMtrSpeed = -255;
        motor.PortMotorDirection = REV;
        motor.PortMotorSpeed = (unsigned char) (-portMtrSpeed);
    }
    //we will mix the throttle signals and the steering signals to make two motor throttle signals
    //we are loosing precision because we are using half of the 255 adc range to make up the full 255 of
the motor
```

```
        //pwm duty cycle
        //return the motor values
        return motor;
}

//This simply prints out what we are sending to the DCH (motor values)
void MAIN_debugDriveMotorOutputsOnLCD(void) {
        LCD_setCursorPosition(1, 12);
        //motorSpeedValues motorSpeeds;
        if(driveMotorSpeeds.StarboardMotorDirection == REV) {
            LCD_writeAsciToLCD('-');
        } else {
            LCD_writeAsciToLCD('+');
        }
        LCD_writeDecimalIntNumberToLCD(driveMotorSpeeds.StarboardMotorSpeed);  //<--

        //writeDecimalNumberToLCD(steeringADCValue);
        LCD_setCursorPosition(1, 5);
        if(driveMotorSpeeds.PortMotorDirection == REV) {
            LCD_writeAsciToLCD('-');
        } else {
            LCD_writeAsciToLCD('+');
        }
        LCD_writeDecimalIntNumberToLCD(driveMotorSpeeds.PortMotorSpeed);   //<---
}

unsigned char MAIN_EncodeOrDecodeByteUsingCyper(unsigned char dataByte) {
        return (dataByte ^ XORCypherKey);
}

void MAIN_AssignFleetAffiliation(unsigned char fleet) {
        FleetAffilation = fleet;
}

void  MAIN_AssignXORCypher(unsigned char cypher) {
        XORCypherKey = cypher;
}

unsigned char MAIN_GetFleetAffiliation(void) {
        return FleetAffilation;
}

unsigned char MAIN_GetXORCypher(void) {
        return XORCypherKey;
}

unsigned char MAIN_GetLastHatSwitchState(void) {
        return lastHatSwitchState;
}

void MAIN_AssignLastHatSwitchState(unsigned char flagState) {
        lastHatSwitchState = flagState;
}

unsigned char MAIN_GetIsHatOnHCFlag(void) {
        return isHatOnHCFlag;
```

```
}

void MAIN_AssignIsHatOnHCFlag(unsigned char flagState) {
	isHatOnHCFlag = flagState;
}

void MAIN_initalizeLCDTextForGamePlay(void) {
	LCD_clearLCD();
	LCD_setCursorPosition(0, 0);
	LCD_writeAsciToLCD('C');
	LCD_writeAsciToLCD('R');
	LCD_writeAsciToLCD('A');
	LCD_writeAsciToLCD('B');
	LCD_writeAsciToLCD(':');
	LCD_writeAsciToLCD(' ');
	LCD_writeAsciToLCD(' ');
	LCD_writeAsciToLCD(' ');
	LCD_writeAsciToLCD('0');
}

void MAIN_initalizeLCDTextForHatOnHC(void) {
	LCD_clearLCD();
	LCD_setCursorPosition(0, 0);
	LCD_writeAsciToLCD('W');
	LCD_writeAsciToLCD('E');
	LCD_writeAsciToLCD('A');
	LCD_writeAsciToLCD('R');
	LCD_writeAsciToLCD(' ');
	LCD_writeAsciToLCD('Y');
	LCD_writeAsciToLCD('O');
	LCD_writeAsciToLCD('U');
	LCD_writeAsciToLCD('R');

	LCD_writeAsciToLCD(' ');
	LCD_writeAsciToLCD('H');
	LCD_writeAsciToLCD('A');
	LCD_writeAsciToLCD('T');
	LCD_writeAsciToLCD(' ');

	LCD_writeAsciToLCD('T');
	LCD_writeAsciToLCD('O');

	LCD_setCursorPosition(1, 0);

	LCD_writeAsciToLCD('D');
	LCD_writeAsciToLCD('R');
	LCD_writeAsciToLCD('I');
	LCD_writeAsciToLCD('V');
	LCD_writeAsciToLCD('E');
	LCD_writeAsciToLCD(' ');
	LCD_writeAsciToLCD('T');
	LCD_writeAsciToLCD('H');
	LCD_writeAsciToLCD('E');

	LCD_writeAsciToLCD(' ');
	LCD_writeAsciToLCD('B');
```

```
        LCD_writeAsciToLCD('O');
        LCD_writeAsciToLCD('A');
        LCD_writeAsciToLCD('T');

        LCD_writeAsciToLCD('!');
        LCD_writeAsciToLCD(' ');
}
```

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


#include "PirateAsync.h"

static usartTxMessage TxUSARTmessageBuffer;  //this is the local buffer where a Transmitting message is stored
static unsigned char sendingUSARTmessageFlag = 0;  //this is a local flag that is raised when a message is being sent

//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

//this function starts sending a usart message
//it raises the sending message flag
//then takes the message contents and stores them in the local buffer
//then it enables async transmit interrupts so that the message can start sending
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;
}

//this is called when the last byte of the message is sent
//it disables the interrupt and lowers the sending flag
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
}

//this function returns the state of the are-we-sending-flag
unsigned char ASYNC_areWeSendingAUSARTmessage(void) {
	return sendingUSARTmessageFlag;
}
```

```
//this function is called from the Interrupt response routine
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++;
	}
}

//this function simply puts a byte into the TXREG  <-- if we have too many nested functions, this one can
be re-encorperated
//by the functions that call it
void ASYNC_sendUSARTbyte(unsigned char dataByteToSend) {
	//do some sort of MAC or line checking here...if we need to
	TXREG = dataByteToSend;
}

//this does the setup for Async communication
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
 * HC 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 for the pin defs

//this is the outgoing message structure
typedef struct{
  unsigned char messageByte[18];
  unsigned char messageIndex;
  unsigned char numberOfBytes;
}usartTxMessage;

/*functions*/
void ASYNC_setupUSART(void);
void ASYNC_sendUSARTmessage(usartTxMessage message);
void ASYNC_finishedSendingUSARTmessage(void);
unsigned char ASYNC_areWeSendingAUSARTmessage(void);
void ASYNC_sendNextUSARTTxMessageByte(void);
void ASYNC_sendUSARTbyte(unsigned char dataByteToSend);

#endif
```

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


#include "PirateLCD.h"  //what does this do?

void LCD_blockingDelay(unsigned int delayTimeH) {
        unsigned char currentTimeH;
        unsigned char lastTimeH;
        unsigned char deltaTime;
        unsigned int timerTime = 0;

        lastTimeH = TMR1H;

        while(timerTime<=delayTimeH) { //loop until we've reached the current time.
            currentTimeH = TMR1H;
            deltaTime = (currentTimeH - lastTimeH);
            timerTime = timerTime + deltaTime;
            lastTimeH = currentTimeH;
        }
}

//this function will initalize the pins that the LCD uses, you probably shouldn't use this...not tested
void LCD_initLCDPortsAndPins(void) {
        CLEARBIT(LCD_DB4_DDR, LCD_DB4_BIT); //output
        CLEARBIT(LCD_DB5_DDR, LCD_DB5_BIT); //output
        CLEARBIT(LCD_DB6_DDR, LCD_DB6_BIT); //output
        CLEARBIT(LCD_DB7_DDR, LCD_DB7_BIT); //output

        CLEARBIT(LCD_RS_DDR, LCD_RS_BIT); //output
        CLEARBIT(LCD_Enable_DDR, LCD_Enable_BIT); //output

        ANSEL = 0;  //NO ADC, all digital pins
        ANSELH = 0;  //NO ADC, all digital pins
}

//clears the entire screen of the LCD and sets the cursor to 0,0
void LCD_clearLCD(void) {
        CLEARBIT(LCD_RS_PORT, LCD_RS_BIT);  //INSTRUCTION TO LCD
        //Clear the screen
        LCD_writeHighNibbleToLCDOutputs(0b00000001); //clear screen
        LCD_pulseLCDEnableLine();
        LCD_writeLowNibbleToLCDOutputs(0b00000001); //clear screen
        LCD_pulseLCDEnableLine();
        //blockingDelay(100);
}
```

```
//sets the cursor home (0,0)
void LCD_moveLCDcursorHome(void) {
	CLEARBIT(LCD_RS_PORT, LCD_RS_BIT);  //INSTRUCTION TO LCD
	//Clear the screen
	LCD_writeHighNibbleToLCDOutputs(0b00000010); //clear screen
	LCD_pulseLCDEnableLine();
	LCD_writeLowNibbleToLCDOutputs(0b00000010); //clear screen
	LCD_pulseLCDEnableLine();
	//blockingDelay(100);
}


//this initalizes the LCD unit itself, not the pins, the pins need to be initalized by you
void LCD_initLCDDisplay(void) {
	LCD_blockingDelay(100);
	//wri/te//CMDToLCD(unsigned char asciChar);

	//STEP 1 = Power Reset
	CLEARBIT(LCD_RS_PORT, LCD_RS_BIT);
	//STEP 2
	LCD_writeHighNibbleToLCDOutputs(0b00100000); //Set the LCD to 4 bit com mode
	LCD_pulseLCDEnableLine();
	LCD_blockingDelay(100);
	//STEP 3
	LCD_writeHighNibbleToLCDOutputs(0b00101000); //Set the LCD to 4 bit com mode, 2 lines
	LCD_pulseLCDEnableLine();
	LCD_writeLowNibbleToLCDOutputs(0b00101000); //Set the LCD to 4 bit com mode, 2 lines
	LCD_pulseLCDEnableLine();
	LCD_blockingDelay(100);
	//STEP 4
	LCD_writeHighNibbleToLCDOutputs(0b00001100); //Display 0n, Cursor off, blinking of cursor off
	LCD_pulseLCDEnableLine();
	LCD_writeLowNibbleToLCDOutputs(0b00001100); //Display 0n, Cursor off, blinking of cursor off
	LCD_pulseLCDEnableLine();
	LCD_blockingDelay(100);
	//STEP 5
	LCD_writeHighNibbleToLCDOutputs(0b00000110); //Set cursor move direction = left to right, and
shifts with new data
	LCD_pulseLCDEnableLine();
	LCD_writeLowNibbleToLCDOutputs(0b00000110); //Set cursor move direction = left to right, and
shifts with new data
	LCD_pulseLCDEnableLine();
	LCD_blockingDelay(100);

	LCD_clearLCD();
	LCD_blockingDelay(100);
	LCD_moveLCDcursorHome();
	//INIT SHOULD BE DONE~!!!
}

void LCD_writeHighNibbleToLCDOutputs(unsigned char dataToWrite) {
	if(CHECKBIT(dataToWrite,4) != 0)    { //then set the output high
	   SETBIT(LCD_DB4_PORT, LCD_DB4_BIT);
	} else {
	   CLEARBIT(LCD_DB4_PORT, LCD_DB4_BIT);
```

```
        }

        if(CHECKBIT(dataToWrite,5) != 0)    { //then set the output high
            SETBIT(LCD_DB5_PORT, LCD_DB5_BIT);
        } else {
            CLEARBIT(LCD_DB5_PORT, LCD_DB5_BIT);
        }

        if(CHECKBIT(dataToWrite,6) != 0) { //then set the output high
            SETBIT(LCD_DB6_PORT, LCD_DB6_BIT);
        } else {
            CLEARBIT(LCD_DB6_PORT, LCD_DB6_BIT);
        }

        if(CHECKBIT(dataToWrite,7) != 0)    { //then set the output high
            SETBIT(LCD_DB7_PORT, LCD_DB7_BIT);
        } else {
            CLEARBIT(LCD_DB7_PORT, LCD_DB7_BIT);
        }
}

void LCD_writeLowNibbleToLCDOutputs(unsigned char dataToWrite) {
        if(CHECKBIT(dataToWrite,0) != 0) { //then set the output high
            SETBIT(LCD_DB4_PORT, LCD_DB4_BIT);
        } else {
            CLEARBIT(LCD_DB4_PORT, LCD_DB4_BIT);
        }

        if(CHECKBIT(dataToWrite,1) != 0) { //then set the output high
            SETBIT(LCD_DB5_PORT, LCD_DB5_BIT);
        } else {
            CLEARBIT(LCD_DB5_PORT, LCD_DB5_BIT);
        }

        if(CHECKBIT(dataToWrite,2) != 0) { //then set the output high
            SETBIT(LCD_DB6_PORT, LCD_DB6_BIT);
        } else {
            CLEARBIT(LCD_DB6_PORT, LCD_DB6_BIT);
        }

        if(CHECKBIT(dataToWrite,3) != 0) { //then set the output high
            SETBIT(LCD_DB7_PORT, LCD_DB7_BIT);
        } else {
            CLEARBIT(LCD_DB7_PORT, LCD_DB7_BIT);
        }
}

//this will write a 0-255 decimal number the the current cursor position.
//it converts it to a series of asci chars
//it doesn't write spaces when the number is shorter than it might be...although it could easily be modified
//to do that.
//this will write a 0-255 decimal number the the current cursor position.
//it converts it to a series of asci chars
//it doesn't write spaces when the number is shorter than it might be...although it could easily be modified
//to do that.
```

```
//I JUST CHANGED THIS TO INPUT AN INT AND NOT A CHAR, ...see what happens...
void LCD_writeDecimalIntNumberToLCD(unsigned int dataInt) {
	unsigned char displayByte[4];
	unsigned char numberOfDigits = 0;
	unsigned int tempData = 0;
	unsigned int tempData2 = 0;

	//convert a char into asci II chars and display it  data = 1 --> ascii 1
	if(dataInt>99) {
	   numberOfDigits = 3;
	} else if(dataInt>9) {
	   numberOfDigits = 2;
	} else {
	   numberOfDigits = 1;
	}

	if(numberOfDigits == 1) {
	   displayByte[0] = 48+dataInt;  //one's place output

	   LCD_writeAsciToLCD(' ');
	   LCD_writeAsciToLCD(' ');
	   //writeAsciToLCD(displayByte[tempData-1]);
	   LCD_writeAsciToLCD(displayByte[0]);
	} else if(numberOfDigits == 2) {
	   tempData = dataInt / 10;
	   displayByte[1] = 48+tempData;  //10's place digit output
	   displayByte[0] = 48+(dataInt-tempData*10);  //one's place output
	   LCD_writeAsciToLCD(' ');
	   LCD_writeAsciToLCD(displayByte[1]);
	   LCD_writeAsciToLCD(displayByte[0]);
	} else if(numberOfDigits == 3) {
	   tempData = dataInt / 100;
	   displayByte[2] = 48+tempData;  //100's place digit output
	   tempData2 = (dataInt-tempData*100) / 10;
	   displayByte[1] = 48+tempData2;  //10's place digit output
	   displayByte[0] = 48+(dataInt -(tempData*100 + tempData2*10) );  //one's place output
	   //writeAsciToLCD(displayByte[tempData-1]);
	   LCD_writeAsciToLCD(displayByte[2]);
	   LCD_writeAsciToLCD(displayByte[1]);
	   LCD_writeAsciToLCD(displayByte[0]);
	}
}

//This sets the row (0 or 1) and column (0-39) of the lcd cursor (which is configed to be invisible)
void LCD_setCursorPosition(unsigned char row, unsigned char column) {
	unsigned char ddramAddress;
	unsigned char cmdByte = 0b10000000;
	//the address rolls around to the lower row when it is higher than 40
	ddramAddress = row*64 + column;

	CLEARBIT(LCD_RS_PORT, LCD_RS_BIT);  //INSTRUCTION TO LCD
	//Clear the screen
	if (ddramAddress<128) { //overflow and error checking
	   cmdByte = cmdByte + ddramAddress;
	   LCD_writeHighNibbleToLCDOutputs(cmdByte); //clear screen
	   LCD_pulseLCDEnableLine();
```

```
            LCD_writeLowNibbleToLCDOutputs(cmdByte); //clear screen
            LCD_pulseLCDEnableLine();
        }
}

//this writes an asci II (?) char to the current position of the LCD cursor
void LCD_writeAsciToLCD(unsigned char asciChar) {
        SETBIT(LCD_RS_PORT, LCD_RS_BIT);
        //write the top four asci bits out to the pins
        LCD_writeHighNibbleToLCDOutputs(asciChar);
        //then pulse the enable line
        LCD_pulseLCDEnableLine();
        LCD_blockingDelay(3);  //10 = 4ms  1 = 0.4 ms
        //then write the lower four asci bits out to the pins
        LCD_writeLowNibbleToLCDOutputs(asciChar);
        //then pulse the enable line again!
        LCD_pulseLCDEnableLine();
        LCD_blockingDelay(3);  //10 = 4ms  1 = 0.4 ms
}

void LCD_pulseLCDEnableLine(void) {
        SETBIT(LCD_Enable_PORT, LCD_Enable_BIT); //high
        LCD_blockingDelay(3); //1ms delay... this is probably way way too long
        CLEARBIT(LCD_Enable_PORT, LCD_Enable_BIT); //low
        LCD_blockingDelay(3); //1ms delay... this is probably way way too long
}
```

```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * HC PIC
 * LCD 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_LCD_H_
#define _PIRATE_LCD_H_

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

/*functions*/
void LCD_blockingDelay(unsigned int delayTimeH);
void LCD_pulseLCDEnableLine(void);
void LCD_initLCDDisplay(void);
void LCD_clearLCD(void);
void LCD_writeAsciToLCD(unsigned char asciChar);
void LCD_writeHighNibbleToLCDOutputs(unsigned char dataToWrite);
void LCD_writeLowNibbleToLCDOutputs(unsigned char dataToWrite);
void LCD_moveLCDcursorHome(void);
void LCD_writeDecimalIntNumberToLCD(unsigned int dataChar);
void LCD_setCursorPosition(unsigned char row, unsigned char column);
void LCD_initLCDPortsAndPins(void);

#endif
```

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


#include "xbee.h"
#include "PirateAsync.h"
#include "PirateLCD.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 NeptuneOpenSeasonFlag = 0;

static unsigned char LastCrabPotAccumulationRate = 0;
static unsigned char LastCrabPotCrabsTranfered = 0;
unsigned char LastFleetReportAccumulationRate = 0;
unsigned char LastFleetReportCrabPotNumber = 0;
unsigned char HaveWeHadANewCrabPotFleetStatusUpdate = 0;

static unsigned int lastReportedCrabOnBoard = 0;
static unsigned char lastReportedWaterLoggedStatus = 0;
static unsigned char newDCHStatusReportFlag = 0;
//static unsigned char newDCHPrivateCrabDataMsgFlag = 0;
static unsigned char lastReportedCrabPotVisited = 0;
static unsigned char lastReportedRateOfCrabs = 0;
static unsigned char lastReportedAmountOfCrabsTaken = 0;
static unsigned char lastReportedXORCypher = 0;
static unsigned char lastReportedImptEvent = 0;
unsigned char lastReportedFleetAssignment = WHITE_FLEET;

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

```
unsigned char XBEE_getLastFleetReportedRate(void) {
	return LastFleetReportAccumulationRate;
}

unsigned char XBEE_getLastFleetReportedPotNumber(void) {
	return LastFleetReportCrabPotNumber;
}

unsigned char XBEE_HaveWeHadANewCrabPotFleetUpdate(void) {
	if(HaveWeHadANewCrabPotFleetStatusUpdate != 0) {
	    //lower the flag if the message has been checked
	    HaveWeHadANewCrabPotFleetStatusUpdate = 0;
	    return 1;
	} else {
	    return 0;
	}
}

unsigned char XBEE_getLastCrabPotAccumulationRate(void) {
	return LastCrabPotAccumulationRate;
}

unsigned char XBEE_getLastAmountOfCrabTransfered(void) {
	return LastCrabPotCrabsTranfered;
}

unsigned char XBEE_getLastReportedFleetAssignment(void) {
	return lastReportedFleetAssignment;
}

void XBEE_SendDriveCommandMessageToDCH(unsigned char portMotDir, unsigned char
portMotDutyCycle, unsigned char starboardMotDir, unsigned char starboardMotDutyCycle) {
	unsigned char driveMessageBuffer[5];
	if(ASYNC_areWeSendingAUSARTmessage() == 0) { //check to see if we are already sending a
message
	    driveMessageBuffer[0] = HCDCH_DRIVE_CMD_MESSAGE;
	    driveMessageBuffer[1] = portMotDir;
	    driveMessageBuffer[2] = portMotDutyCycle;
	    driveMessageBuffer[3] = starboardMotDir;
	    driveMessageBuffer[4] = starboardMotDutyCycle;
	    ASYNC_sendUSARTmessage(XBEE_populateRadioMessage(DCH_RADIO_ADDRESS_MSB,
DCH_RADIO_ADDRESS_LSB, driveMessageBuffer, 5));
	}
}

usartTxMessage XBEE_populateRadioMessage(unsigned char MsgAddrHigh, unsigned char
MsgAddrLow, unsigned char MsgData[], unsigned char MsgDataLength) {
	usartTxMessage myMessage;
	unsigned char MsgIndex = 0;
	unsigned char MsgLengthHighByte = 0;
	unsigned char MsgLengthLowByte = 0;
	unsigned char frameID = 0;
	unsigned char checksumValue = 0;

	//frameID = TMR1H;
```

```
        frameID = 0x26;
        //MESSAGE LENGTH = FRAME DATA ONLY (NO CHECKSUM)
        MsgLengthLowByte = 5 + MsgDataLength;  //I'm going to say that this will be less than 255
        myMessage.numberOfBytes = MsgDataLength + 9;
        myMessage.messageByte[0] = RADIO_MESSAGE_START_DELIMETER;  //first byte is the start
byte
        myMessage.messageByte[1] = MsgLengthHighByte; //length
        myMessage.messageByte[2] = MsgLengthLowByte;  //length
        myMessage.messageByte[3] = RADIO_MESSAGE_SEND_API_IDENTIFIER;
        myMessage.messageByte[4] = frameID;
        myMessage.messageByte[5] = MsgAddrHigh;
        myMessage.messageByte[6] = MsgAddrLow;
        myMessage.messageByte[7] = 0x00;  //OPTIONS BYTE  (default)
        //messageData
        for(MsgIndex = 0; MsgIndex<MsgDataLength; MsgIndex++){
            myMessage.messageByte[8+MsgIndex] = MsgData[MsgIndex];
        }

        //checksum
        for(MsgIndex = 3; MsgIndex<3+MsgLengthLowByte; MsgIndex++) {
            checksumValue += myMessage.messageByte[MsgIndex];
        }

        checksumValue = 0xFF - checksumValue;
        myMessage.messageByte[8+MsgDataLength] = checksumValue;  //checksum
        return myMessage;
}

/* 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_getLastWaterLoggedStatusFromDCH(void) {
        return lastReportedWaterLoggedStatus;
}

unsigned int XBEE_getLastCrabCountFromDCH(void) {
        return lastReportedCrabOnBoard;
}

unsigned char XBEE_getLastCrabPotVisited(void) {
        return lastReportedCrabPotVisited;
}
```

```
unsigned char XBEE_getlastReportedRateOfCrabs(void) {
	return lastReportedRateOfCrabs;
}

unsigned char XBEE_getLastReportedAmountOfCrabsTaken(void) {
	return lastReportedAmountOfCrabsTaken;
}

unsigned char XBEE_getLastReportedXORCypher(void) {
	return lastReportedXORCypher;
}

unsigned char XBEE_getLastReportedImptEvent(void) {
	return lastReportedImptEvent;
}

unsigned char XBEE_HasNewDCHStatusMessageBeenReceived(void) {
	if(newDCHStatusReportFlag != 0) {
	    //lower the flag if the message has been checked
	    newDCHStatusReportFlag = 0;
	    return 1;
	} else {
	    return 0;
	}
}

unsigned char XBEE_CheckNeptuneOpenSeasonFlag(void) {
	if(NeptuneOpenSeasonFlag != 0) {
	    //lower the flag if the message has been checked
	    NeptuneOpenSeasonFlag = 0;
	    return 1;
	} else {
	    return 0;
	}
}

/* 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 things
                        //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?
            //setCursorPosition(0,2);
            //writeAsciToLCD('V'); //visible
            if(checkSum != 0xFF) {
                        //setCursorPosition(0,3);
                        //writeAsciToLCD('B');  //not visible

                        //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 ultimatly, i'll set flags for now
                        //first, what type of message is it?
                        //look at source address
                        if(bufferContents[INDEX_INCOMMING_SOURCE_ADDR_MSB] ==
NEPTUNE_RADIO_ADDR_MSB && bufferContents[INDEX_INCOMMING_SOURCE_ADDR_LSB]
== NEPTUNE_RADIO_ADDR_LSB) {
                                    //if the source address of the radio message matches neptunes address
                                    //is the message private?
                                    if((bufferContents[INDEX_INCOMMING_OPTIONS_BYTE] & 0x02) == 0) {
//this seems to work
                                                //it is a private message, therefor it is either a crab pot respoce, a repair
dock responce, or a drydock responce
                                                //Should never happen
                                    } else { //or broadcast?
                                                //it is most likely open-season, is it?
                                                if(msgFrameLength-5 == 3){
                                                            //setCursorPosition(0,7);
                                                    //writeAsciToLCD('M');  //not visible
                                                            //if it has three bytes in the data packet, it's even more likely to
be open-season

        if(bufferContents[INDEX_INCOMMING_START_OF_DATA] ==
NEPTUNE_OPEN_SEASON_BYTE_ONE &&

bufferContents[INDEX_INCOMMING_START_OF_DATA+1] ==
NEPTUNE_OPEN_SEASON_BYTE_TWO &&

bufferContents[INDEX_INCOMMING_START_OF_DATA+2] ==
NEPTUNE_OPEN_SEASON_BYTE_THREE) {
                                                                        //THERE WAS AN OPEN SEASON COMMAND
FROM NEPTUNE!!
                                                                        //set the neptune open season flag (when is this
lowered?  on reset?  anyways, it needs to be lowered)
                                                                        NeptuneOpenSeasonFlag = 1;
                                                            }
                                                }
```

```
                        }
                    } else if(bufferContents[INDEX_INCOMMING_SOURCE_ADDR_MSB] ==
DCH_RADIO_ADDRESS_MSB && bufferContents[INDEX_INCOMMING_SOURCE_ADDR_LSB]
== DCH_RADIO_ADDRESS_LSB) {
                        //if the source address of the radio message matches or DCH's address
                        //is the message private?
                        LCD_setCursorPosition(1, 0);
                    LCD_writeAsciToLCD('1');   //if we ever see this it's bad
                        if((bufferContents[INDEX_INCOMMING_OPTIONS_BYTE] & 0x02) == 0) {
//private message
                            LCD_setCursorPosition(1, 0);
                            LCD_writeAsciToLCD('2');   //if we ever see this it's bad   //it got here
fine, but not below
                            //it is a private message meaning it could be a status message, a private
crab pot message(for testing), or a team pairing message, ie, has security key
                            if(bufferContents[INDEX_INCOMMING_START_OF_DATA] ==
DCHHC_STATUS_MESSAGE) {  //the problem is at this if statement, check
                                //i've removed the above if statement in an attempt to
isolate the source of the problem
                                //the problem is that the DCH sends status updates
correctly to the HC
                                //but the HC doesn't update the LCD or indicate that
it has gotten some of them
                                //some of them work, but not all of them...perhaps
there is noise on the line?

                                //is the problem on the HC side or the dch side?
                                LCD_setCursorPosition(1, 0);
                                LCD_writeAsciToLCD('3');   //if we ever see this it's bad
                                //we have received a status message from the DCH
                                //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];
                                lastReportedCrabPotVisited =
bufferContents[INDEX_INCOMMING_START_OF_DATA+4];
                                lastReportedRateOfCrabs =
bufferContents[INDEX_INCOMMING_START_OF_DATA+5];
                                lastReportedAmountOfCrabsTaken =
bufferContents[INDEX_INCOMMING_START_OF_DATA+6];
                                lastReportedImptEvent =
bufferContents[INDEX_INCOMMING_START_OF_DATA+7];
                                lastReportedXORCypher =
bufferContents[INDEX_INCOMMING_START_OF_DATA+8];
                                lastReportedFleetAssignment =
bufferContents[INDEX_INCOMMING_START_OF_DATA+9];
                                //raise the flag
                                newDCHStatusReportFlag = 1;  //set the status message flag
to let the event checker know we've gotten a status message
                            }
                        } else { //or broadcast?
                            //this would mean that it is either a boat pairing command (ignore) or a
team encrypted crab pot update (do same thing as below)
                            if(bufferContents[INDEX_INCOMMING_START_OF_DATA] ==
```

```
MAIN_GetFleetAffiliation()) { // && BOAT_STATUS_getBoatStatus().myFleet != WHITE)
                                        //if the incomming broadcast message matches our fleet
                                        //it is someone broadcasting pot information
                                        //if it is a fleetmate asking for the XOR cypher, we should give
it to him, or at least let the event
                                        //checker know so it can give it it him
                                        //otherwise, it is probably a fleet crabpod data broadcast, in
which case we need can ignore it.  Our
                                        //HC will listen to it and update the LCD screen appropratly

        if(bufferContents[INDEX_INCOMMING_START_OF_DATA+1] ==
FLEET_MSG_CRAB_DATA) {
                                                //it is a fleet crab data broadcast, it's encoded
                                                //decode the fleet broadcast and store the data
                                                //this decodes the reported (by the fleet) crab pot
number  (-0xA0 gives us the raw number)
                                                LastFleetReportCrabPotNumber =
MAIN_EncodeOrDecodeByteUsingCyper(bufferContents[INDEX_INCOMMING_START_OF_DATA+2
])-CRAB_POT_0;
//MAIN_EncodeOrDecodeByteUsingCyper(bufferContents[INDEX_INCOMMING_START_OF_DATA+
2], MAIN_GetXORCypher()) - 0xA0;
                                                //this decodes the last reported accumulation rate
                                                LastFleetReportAccumulationRate =
MAIN_EncodeOrDecodeByteUsingCyper(bufferContents[INDEX_INCOMMING_START_OF_DATA+3
]);
//MAIN_EncodeOrDecodeByteUsingCyper(bufferContents[INDEX_INCOMMING_START_OF_DATA+
3], MAIN_GetXORCypher());
                                                //unsigned char dataToSendToFleet[4];
                                                //populate a local array
                                                //dataToSendToFleet[0] =
BOAT_STATUS_getBoatStatus().myFleet;
                                                //dataToSendToFleet[1] =
FLEET_MSG_CRAB_DATA;
                                                //dataToSendToFleet[2] =
MAIN_EncodeOrDecodeByteUsingCyper(crabPotNumber);
                                                //dataToSendToFleet[3] =
MAIN_EncodeOrDecodeByteUsingCyper(crabPotRate);
                                                //set a flag to let the event checker know that this has
happened
                                                HaveWeHadANewCrabPotFleetStatusUpdate = 1;
                                        }
                                }
                        }
                } else {
                        //if this is the HC, it will be neccissary to have other provisions
                        //is the message private? (should never happen)
                        //or broadcast?  (team crab pot data)
                        //this is team pot data
                        if((bufferContents[INDEX_INCOMMING_OPTIONS_BYTE] & 0x02) == 0) {
//private message
                                //this shouldn't happen
                        } else {
                                //this is probably a team broadcast....
                                if(bufferContents[INDEX_INCOMMING_START_OF_DATA] ==
MAIN_GetFleetAffiliation()) { // && BOAT_STATUS_getBoatStatus().myFleet != WHITE)
                                        //if the incomming broadcast message matches our fleet
```

```
                                        //it is someone broadcasting pot information
                                        //if it is a fleetmate asking for the XOR cypher, we should give
it to him, or at least let the event
                                        //checker know so it can give it it him
                                        //otherwise, it is probably a fleet crabpod data broadcast, in
which case we need can ignore it.  Our
                                        //HC will listen to it and update the LCD screen appropratly

      if(bufferContents[INDEX_INCOMMING_START_OF_DATA+1] ==
FLEET_MSG_CRAB_DATA) {
                                                //it is a fleet crab data broadcast, it's encoded
                                                //decode the fleet broadcast and store the data
                                                //this decodes the reported (by the fleet) crab pot
number  (-0xA0 gives us the raw number)
                                                LastFleetReportCrabPotNumber =
MAIN_EncodeOrDecodeByteUsingCyper(bufferContents[INDEX_INCOMMING_START_OF_DATA+2
])-CRAB_POT_0;
//MAIN_EncodeOrDecodeByteUsingCyper(bufferContents[INDEX_INCOMMING_START_OF_DATA+
2], MAIN_GetXORCypher()) - 0xA0;
                                                //this decodes the last reported accumulation rate
                                                LastFleetReportAccumulationRate =
MAIN_EncodeOrDecodeByteUsingCyper(bufferContents[INDEX_INCOMMING_START_OF_DATA+3
]);
//MAIN_EncodeOrDecodeByteUsingCyper(bufferContents[INDEX_INCOMMING_START_OF_DATA+
3], MAIN_GetXORCypher());
                                                //unsigned char dataToSendToFleet[4];
                                                //populate a local array
                                                //dataToSendToFleet[0] =
BOAT_STATUS_getBoatStatus().myFleet;
                                                //dataToSendToFleet[1] =
FLEET_MSG_CRAB_DATA;
                                                //dataToSendToFleet[2] =
MAIN_EncodeOrDecodeByteUsingCyper(crabPotNumber);
                                                //dataToSendToFleet[3] =
MAIN_EncodeOrDecodeByteUsingCyper(crabPotRate);
                                                //set a flag to let the event checker know that this has
happened
                                                HaveWeHadANewCrabPotFleetStatusUpdate = 1;
                                        }
                                }
                        }
                }
        } //end checksum check else
      } //end is the message API identifier correct for an incomming radio message?
}
```

```
/*
 * Team Seconds from Disaster
 * 5.25.10
 * ME218c
 * HC 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 HCDCH_DRIVE_CMD_MESSAGE 0xF1
#define DCHHC_STATUS_MESSAGE 0xF2
#define DCHHC_SECURITY_KEY_MESSAGE 0xF4
#define DCHHC_PRIVATE_CRAB_MESSAGE 0xF3

#define FLEET_MSG_TYPE_IM_ON_TEAM 0x01
#define FLEET_MSG_TYPE_THE_CYPHER_IS 0x02
#define FLEET_MSG_CRAB_DATA 0x03

#define RX_BUFFER_SIZE 24

//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
```

```
//important neptune data defines
#define NEPTUNE_OPEN_SEASON_BYTE_ONE 0x47
#define NEPTUNE_OPEN_SEASON_BYTE_TWO 0x6F
#define NEPTUNE_OPEN_SEASON_BYTE_THREE 0x21

#define NEPTUNE_REQUEST_BYTE 0x66

#define NEPTUNE_RESP_INVALID_KEY 253
#define NEPTUNE_RESP_SHIP_UNLOADED 254
#define NEPTUNE_RESP_SHIP_REPAIRED 255

//types of neptune responce that the main program can see
#define NEP_RESP_WAITING_FOR_RESPONSE 0
#define NEP_RESP_CRAB_POT_DATA 1
#define NEP_RESP_INVALID_KEY 2
#define NEP_RESP_SHIP_UNLOADED 3
#define NEP_RESP_SHIP_REPAIRED 4
#define NEP_RESP_COMPLEMENT_MIS_MATCH 5

/*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);
unsigned char XBEE_getLastCrabPotAccumulationRate(void);
unsigned char XBEE_getLastAmountOfCrabTransfered(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);
unsigned char XBEE_CheckNeptuneOpenSeasonFlag(void);
void XBEE_SendNextMessageByte(void);
void XBEE_ReadIncommingAsyncByte(unsigned char rxByte);
void XBEE_InterperateReceivedMessage(void);
void XBEE_SendDriveCommandMessageToDCH(unsigned char portMotDir, unsigned char
portMotDutyCycle, unsigned char starboardMotDir, unsigned char starboardMotDutyCycle);
unsigned char XBEE_getLastWaterLoggedStatusFromDCH(void);
unsigned int XBEE_getLastCrabCountFromDCH(void);
unsigned char XBEE_HasNewDCHStatusMessageBeenReceived(void);
unsigned char XBEE_HasNewDCHPrivateCrabMsgBeenReceived(void);
unsigned char XBEE_getLastCrabPotVisited(void);
unsigned char XBEE_getlastReportedRateOfCrabs(void);
unsigned char XBEE_getLastReportedAmountOfCrabsTaken(void);
unsigned char XBEE_getLastReportedXORCypher(void);
unsigned char XBEE_getLastReportedImptEvent(void);
unsigned char XBEE_getLastFleetReportedRate(void);
unsigned char XBEE_getLastFleetReportedPotNumber(void);
unsigned char XBEE_HaveWeHadANewCrabPotFleetUpdate(void);
```

```
unsigned char XBEE_getLastReportedFleetAssignment(void);

#endif
```