```
/*
* ZIGBEE.c
*
* Created: 10/19/2012 9:08:22 AM
*  Author: starkca
*/

#define F_CPU 16000000UL
#define ZIGBEE_PIN PD4
#define LCD_PIN PD5
#define SRCADDR 42
#define DSTADDR 0
#define TOLCD 0
#define TOZIGBEE 1

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>

#include <stdio.h>
#include <stdbool.h>

#include "uart.h"
#include "adc.h"

// Should the Heart be toggled
volatile bool pulseHrt = false;
// Is heart currently being displayed or not
volatile bool heart = false;
// Heart is in process of changing
volatile bool hrtCng = false;
// Can the system goto sleep?
volatile bool canSleep = false;
// Should the temperature be updated?
volatile bool updateTemp = false;
// Is the system off?
volatile bool systemOff = false;
// Toggle system power status
volatile bool togglePower = false;

// Counter to keep track of how long system has been inactive
volatile uint8_t sleepCnt = 0;
// Destination Address
volatile uint8_t destAddr = 0;


/**************************************************************************/
/* Pulses the heart for the required heart beat.                          */
/* Displays an H to signify the heart.                                    */
/*                                                                        */
/* PARAMETERS:                                                            */
/*  - VOID                                                                */
```

```c
/*                                                                      */
/* RETURNS:                                                             */
/*  - VOID                                                              */
/************************************************************************/
void pulseHeart(void)
{
    // Going to be printing messages, disable interrupts
    cli();

    // Set flag to inform system the heart is changing
    hrtCng = true;

    // The heart is current present
    if(heart)
    {
        // Goto heart position
        hrt;
        // Clear the heart
        printf(" ");
    }
    // Heart is not present
    else
    {
        // Goto heart position
        hrt;
        // Display the heart with alarm (Pretty funny with African American Heart Monitor
        // As seen on Family Guy)
        //      printf("H\a");
        // Display the heart without sound
        printf("H");
    }

    // Clear heart change flag
    hrtCng = false;
    // Toggle heart flag
    heart = !heart;
    // Clear toggle heart flag
    pulseHrt = false;

    // Enable Interrupts
    sei();
}

/************************************************************************/
/* Initialize Timer Counter 1 to track when the heart beat should pulse */
/*                                                                      */
/* PARAMETERS:                                                          */
/*  - VOID                                                              */
/*                                                                      */
/* RETURNS:                                                             */
/*  - VOID                                                              */
/************************************************************************/
void initHeart(void)
```

```c
{
    // Clear the counter
    TCCR1A = 0;

    // Set overflow value to get 1/2 interrupt
    OCR1A = 7810;

    // CTC Mode
    TCCR1B |= 1 << WGM12;

    // 1024 Prescale
    TCCR1B |= (1 << CS10) | (1 << CS12);

    // Enable Interrupt
    TIMSK |= 1 << OCIE1A;
}

/*************************************************************************/
/* Initializes the push button to control the functionality on or off   */
/*************************************************************************/
void initButton(void)
{
    // PD2 Input
    DDRD |= 1 << PD2;
    // Enable Pull-Up Resistor
    PORTD |= 1 << PD2;

    // Trigger falling edge Interrupt
    MCUCR |= 1 << ISC10 | 1 << ISC00;
    // Enable INT0 Interrupts
    GICR |= 1 << INT0;
}

/*************************************************************************/
/* Toggles the pins that control the filter gates on where the          */
/* USART data will be going to, either the XBee or LCD. Filter gate is  */
/* active low, therefore the destination pin is driven low, and other   */
/* pin is driven high.                                                  */
/*************************************************************************/
void switchFunctions(uint8_t function)
{
    if(function == TOZIGBEE)
    {
        PORTD |= 1 << LCD_PIN;
        PORTD &= ~(1 << ZIGBEE_PIN);
    }
    else if(function == TOLCD)
    {
        PORTD |= 1 << ZIGBEE_PIN;
        PORTD &= ~(1 << LCD_PIN);
    }
    else
    {
        PORTD |= (1 << ZIGBEE_PIN) | (1 << LCD_PIN);
    }
}
/**************************************************************************/
/* Main entry point into system                                          */
/* Reads a temperature using the ATMEGA32 ADC subsystem, displays on     */
/* serially connected LCD and sends data via connected XBee Antenna      */
/**************************************************************************/
int main(void)
{
    // Set gate Bits as output
    DDRD |= (1 << ZIGBEE_PIN) | (1 << LCD_PIN);
    switchFunctions(TOLCD);

    uart_init();
    initButton();
    initHeart();
    initADC();

    updateTemp = true;

    clrScr;
    printf("CE3200 ZIGBEE 1.0\nWelcome");

    _delay_ms(1000);

    sei();

    while(1)
    {
        // System says it's time to pulse the heart
        if(pulseHrt && !hrtCng)
            pulseHeart();

        if(updateTemp)
        {
            switchFunctions(TOLCD);

            updateTemp = false;
            // Will start the conversion then wait until conversion is done
            adcStart();
            // Get the value generated by ADC
            uint8_t adcVal = getADCL();

            uint8_t adcValH = getADCH();

            // Fahrenheit = adcVal / 2
            uint8_t tempF = adcVal >> 1;

            // Celsius = (F - 32) / 1.8
            // 1.8 can be rounded to 2 for sake of performance
            uint8_t tempC = (tempF - 32) >> 1;
            clrScr;
            printf("%u F\t%u C",tempF,tempC);
            pwr;
            printf("ON");

            // Set PORTB as input
            DDRB = 0x00;
            // Read in destination address
            destAddr = PINB;

            // Allow the final bits to move
            _delay_ms(2);

            switchFunctions(TOZIGBEE);

            // Allow the switch signal to settle
            _delay_ms(2);

            // Send Source Address
            uart_putc(SRCADDR);
            // Send ADCH
            uart_putc(tempC);
            // Send ADCL
            uart_putc(tempF);

            // Allow the final bits to move
            _delay_ms(10);

            switchFunctions(TOLCD);
        }

        if(togglePower)
        {
            togglePower = false;

            _delay_ms(100);
            while(!(PIND & 1 << PD2));
            _delay_ms(100);

            // System is on, need to turn off
            if(!systemOff)
            {
                // Turn off Timer/Counter 1 as this is where all flag sets happen
                TCCR1B &= ~((1 << CS12) | (1 << CS11) | (1 << CS10));
                pwr;
                printf("OFF");
                canSleep = true;
                systemOff = true;
            }
            // System is off, need to turn on
            else
            {
                // Re-Enable Timer/Counter 1 as this is where all flag sets happen
```

```c
                TCCR1B |= (1 << CS12) | (1 << CS10);
                printf("ON");
                canSleep = false;
                systemOff = false;
            }
        }

        // Set sleep mode to idle
        set_sleep_mode(SLEEP_MODE_IDLE);

        cli();
        if(canSleep)
        {
            sleep;
            printf("SLEEPING");
            sleep_enable();
            sei();
            sleep_cpu();
            sleep_disable();
        }

        sei();
    }
    return 0;
}

/*************************************************************************/
/* Toggles functionality of system. Powering down will disable ability  */
/* for system to update temp and enter any other mode.                   */
/*************************************************************************/
ISR(INT0_vect)
{
    togglePower = true;
    updateTemp = true;
}

/*************************************************************************/
/* Interrupt for Timer Counter for Heart Beat and responsible for        */
/* tracking when the system can sleep and when it should wake up.        */
/*************************************************************************/
ISR(TIMER1_COMPA_vect)
{
    // System is currently sleeping, but hasn't slept enough yet
    if(sleepCnt < 240 && canSleep)
        // Increment counter for sleep time
        sleepCnt++;
    // System is currently sleeping, has slept enough
    else if(sleepCnt >= 240 && canSleep)
    {
        // Reset sleep time
        sleepCnt = 0;
        // Inform system it cannot sleep
        canSleep = false;
        // Inform system we want to refresh the temp
        updateTemp = true;
    }
    // System is currently running, but has not sat idle enough yet
    else if(sleepCnt < 10 && !canSleep)
    {
        // Increment idle count
        sleepCnt++;

        // Heart is currently not changing
        if(!hrtCng)
            // Set flag, system will get right on that
            pulseHrt = true;
    }
    // System is currently running, has been idle long enough, time to enter sleep
    else
    {
        // Reset idle count
        sleepCnt = 0;
        // Inform system it should enter sleep
        canSleep = true;
    }
}
```

```c
/*
 * ADC.h
 *
 * Created: 10/19/2012 12:05:01 PM
 *  Author: starkca
 */


#ifndef ADC_H_
#define ADC_H_

void initADC(void);

void adcStart(void);

uint8_t getADCL(void);

uint8_t getADCH(void);

#endif /* ADC_H_ */
```

```
/*
 * ADC.c
 *
 * Created: 10/19/2012 11:59:13 AM
 *  Author: starkca
 */

#include <avr/io.h>

#include "adc.h"

void initADC(void)
{
    // PORTA all inputs
    DDRA = 0x00;

    // AVCC as reference
    ADMUX |= (1 << REFS0);

    // Enable ADC, set prescaler to clk/128
    ADCSRA |= (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
}

void adcStart(void)
{
    ADCSRA |= 1 << ADSC;

    while(!(ADCSRA & (1<<ADIF)));

    ADCSRA |= 1 << ADIF;
}

uint8_t getADCL(void)
{
    return ADCL;
}

uint8_t getADCH(void)
{
    return ADCH;
}
```

```
/* FILE:   uart.h
 * AUTHOR: Casey Stark <starkca@msoe.edu>
 * COURSE: CE3910
 * DATE:   3/14/12
 *
 * PURPOSE: Header file for UART API
 *          Contains declerations for
 *          functions available with this
 *          API and and constants required
 */

#ifndef uart_h
#define uart_h

#include <stdint.h>
#include <stdio.h>

// Size for UART buffer
#define MAX_BUFFER_SIZE 50

#define clrScr printf("\e[2J \e[H")
#define home printf("\e[H")
#define clrLn printf("\e[K")
#define hrt printf("\e[1;32f")
#define sleep printf("\e[2;0f")
#define pwr printf("\e[2;10f")

/*
*   PURPOSE: Initializes UART Functionality for
*               AtMega32. Enables functionality for
*               C's stdio functions as well.
*
*   PARAMETERS: None
*
*   RETURNS: None
*/
void uart_init(void);

/*
*   PURPOSE: Grabs char from UDR, if char is return char, reset
*               buffer and take appropriate actions, otherwise
*               echo all printable chars back. If buffer becomes
*               full, send a beep as a warning. Backspace is also
*               implemented appropriatly.
*
*   PARAMETERS: None
*
*   RETURNS: Character that was processed
*/
char uart_getc(void);

/*
*   PURPOSE: Add char to queue to be sent out.
*               If char is '\n', send also '\r'
*
*   PARAMETERS: char c: character to put transmited via
*                       serial connection
*
*   RETURN: None
*/
void uart_putc(char c);

/*
*   PURPOSE: Obtains the value located at regAddress
*               the prints it via stdio and returns
*               given value.
*
*   PARAMETERS: uint16_t regAddress: Address to
*                       collect data from
*
*   RETURNS:    Data at given regAddress
*/
uint8_t readIO(uint16_t regAddress);

/*
*   PURPOSE: Writes data to regAddress
*
*   PARAMETERS: uint16_t regAddress: Address of IO port
*                   to write data to.
*               uint8_t data: Data to be writen to
*                   regAddress
*
*   RETURN: None
*/
void writeIO(uint16_t regAddress,uint8_t data);

#endif
```

```c
/* FILE:   uart.c
 * AUTHOR: Casey Stark <starkca@msoe.edu>
 * COURSE: CE3910
 * DATE:   3/14/12
 *
 * PURPOSE: This file contains functions
 *          that are required for
 *          UART communication.
 *          Functions include an
 *          Initializer, putc, and
 *          getc methods.
 */

#include "uart.h"
#include <avr/io.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

// Value to tell UART operations the clock
// speed and desired BAUD Rate.
#define UBRR_DEFAULT 416
#define UBRR_WRITE 103

#define LCD_BAUD_H 0x00
#define LCD_BAUD_L 0x67

volatile char RX_BUFF[MAX_BUFFER_SIZE];
volatile char* rxptr;
volatile char* cptr;

// Create FILE that allows for UART to take over C IO functions
FILE uart_str = FDEV_SETUP_STREAM(uart_putc,uart_getc,_FDEV_SETUP_RW);

/*
PURPOSE:    Initializes UART functionality for AtMega32. Takes over C's stdio
            functions.
PARAMETERS: VOID
RETURNS:    VOID
*/
void uart_init()
{
    UBRRH = LCD_BAUD_H;
    UBRRL = LCD_BAUD_L;

    UCSRA = 0;

    // Transmit and Receive
    UCSRB = (1<<TXEN)|(1<<RXEN) |(1<<RXCIE);

    // synchronous operation, 8-bit char size
    UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
    // initialize pointers to 0
    rxptr = 0;
    cptr = 0;

    // Finish up C IO integration
    stdout=stdin=&uart_str;

    return;
}

/*
PURPOSE:    Add char to queue to be sent out.
            If char is '\n', send also '\r'
PARAMETERS: char c: character to be transmitted
RETURN:     VOID
*/
void uart_putc(char c)
{


    // add the char to the UDR
    UDR = c;

    // if the char is a newline, also send return
    if(c == '\n')
    {
        uart_putc('\r');
    }

    // wait here until the UDR is empty
    while(!(UCSRA&(1<<UDRE)));

    return;
}

/*
PURPOSE:    Grabs char from UDR, if char is return char, reset
            buffer and take appropriate actions, otherwise
            echo all printable chars back. If buffer becomes
            full, send a beep as a warning. Backspace is also
            implemented appropriately.
PARAMETERS: VOID
RETURNS:    char: Processed Character
*/
char uart_getc(void)
{
    char c;
    // Start of new line?
    if(rxptr == 0)
    {
        // Write buffer = start of buffer
        for(cptr = RX_BUFF;;)
        {
            // poll for new character
            while(!(UCSRA&(1<<RXC)));
            c = UDR;
            // if the char is a return, replace with newline,
            // increment the pointer, send the newline, reset
            // the read pointer and break the loop
            if(c=='\r')
            {
                c = '\n';
                *cptr = c;
                cptr++;

                uart_putc(c);
                rxptr = RX_BUFF;
                break;
            }
            // if char is printable
            if((c >= ' ') && (c < 0x7F))
            {
                // if the buffer is full, send a beep to the terminal
                if(cptr == RX_BUFF + MAX_BUFFER_SIZE - 2)
                {
                    uart_putc('\a'); // beep
                }
                // otherwise set the char, increment the pointer, and send it
                else
                {
                    *cptr = c;
                    cptr++;
                    uart_putc(c);
                }

            }
            // if backspace or delete
            if((c == 0x08) || (c == 0x7F))
            {
                // if the write pointer is not at the start of the buffer
                if(cptr > RX_BUFF)
                {
                    uart_putc(0x08); // send backspace
                    uart_putc(' ');  // send space to overwrite previous char
                    uart_putc(0x08); // send backspace
                    cptr--; // decrement the buffer write pointer
                }
            }
        }
    }
    // get the character
    c = *rxptr;
    // increment the read pointer
    rxptr++;
    // if the char was a newline, reset the read pointer to 0
    if(c == '\n')
    {
        rxptr = 0;
    }
    // return the char
    return c;
}

/*
*   PURPOSE: Obtains the value located at regAddress
*                the prints it via stdio and returns
*                given value.
*
*   PARAMETERS: uint16_t regAddress: Address to
*                        collect data from
*
*   RETURNS:    Data at given regAddress
*/
uint8_t readIO(uint16_t regAddress)
{
    uint8_t regData = *(volatile uint8_t*) regAddress;
    printf("Register %u contains %u\n\n", regAddress, regData);
    return regData;
}

/*
*   PURPOSE: Writes data to regAddress
*
*   PARAMETERS: uint16_t regAddress: Address of IO port
*                    to write data to.
*                uint8_t data: Data to be writen to
*                    regAddress
*
*   RETURN: None
*/
void writeIO(uint16_t regAddress,uint8_t data)
{
    if(data <= 0xFF)
    {
        volatile uint8_t* regData = (uint8_t*) regAddress;
        *regData = data;
        if(*regData == data)
        {
            printf("Value %u now resides in %u\n", *regData, regAddress);
        }
        else
        {
            printf("Something Failed");
        }
    }
    else
    {
        printf("How Big Do You Think My Data Capacity Is? Enter A Smaller Number For Data.");
    }
    printf("\n");
    return;
}
```