Week 2 milestone

You must demonstrate that your completed board is functional. Do this by executing a simple "echo back" application that simply reads a character from the serial input and echoes it back to the serial output. Use a terminal emulator such as putty or realterm for your demonstration.

The Serial API planning begins. You should begin planning the development of the serial API. The functions needed by the serial API are:

Function	Description
uartInit()	Initializes uart baud rate and enables transmitter and receiver
uartPutc()	Writes a character to serial output device
uartGetc()	Gets a character from the serial device. Characters that are entered by the user must be echoed back to the serial monitor. This function must allow editing of the line entered by the user as discussed in lecture.

The header file for this API should look something like:

```
#ifndef uart_h
#define uart_h

#include <avr/io.h>
#include <inttypes.h>
#include <stdint.h>


/* Baudrate settings. Refer to datasheet for baud rate error.
   Note also maximun baud rate.
   br = baudrate, fosc = clock frequency in megahertzs */

#define uart_BAUDRATE(br, fosc) (fosc*125000/br-1) // assumes U2X = 1
#define SYSCLK 16000000L
#define RX_BUFSIZE 80

/*****************************************
 Function prototypes
*****************************************/

/*****************************************************************************
        Provides: Initializes uart device. Use uart_BAUDRATE macro for argument or
        consult datasheet for correct value.

        Inputs: ubrrl value

        Returns: none
```

```
*******************************************************************************/
void uart_init(unsigned char baud_divider);

/*******************************************************************************
        Provides: Transmit one character. No buffering, waits until previous character
        transmitted.

        Inputs: Character to be transmitted

        Returns: none

*******************************************************************************/
void uart_putc(char c);

/*******************************************************************************
        Provides: Receive one character.
 * This features a simple line-editor that allows to delete and
 * re-edit the characters entered, until either CR or NL is entered.
 * Printable characters entered will be echoed using uart_putc().
 *
 * Editing characters:
 *
 *  \b (BS) or \177 (DEL) delete the previous character
 *  \t will be replaced by a single space
 *
 * All other control characters will be ignored.
 *
 * The internal line buffer is RX_BUFSIZE (80) characters long, which
 * includes the terminating \n (but no terminating \0).  If the buffer
 * is full (i. e., at RX_BUFSIZE-1 characters in order to keep space
 * for the trailing \n), any further input attempts will send a \a to
 * uart_putc() (BEL character), although line editing is still
 * allowed.
 *
 * Successive calls to uart_getchar() will be satisfied from the
 * internal buffer until that buffer is emptied again.

        Inputs: none

        Returns: character entered by user

*******************************************************************************/
char uart_getc(void);
```

```
#endif
```

The functions uartPutc() and uartGetc() will be used as the stdout and stdin functions thus allowing use of the standard C library as discussed in lecture.

This milestone will include the ability to parse an input command entered by the user from the serial input device using the sscanf function. The commands that will be used to demonstrate the serial API will be rdIO and wrIO. These commands will appear as follows:

```
rdIO regaddress
wrIO regaddress regdata
```

These commands will allow the user to read from and write to IO ports on the Atmega32. Each IO port has a unique address (found in the Atmega32 reference guide). The addresses are 16-bits and must be *entered by the user in hexadecimal*. The rdIO should display the results of the read in the form:

IOregister = xxxx IOdata = yy, where xxxx is the IO address in hex and yy is the data read from the port.

Additional functions needed for the demonstration of the milestone are:

```
uint8_t readIO(uint16_t regAddress);
void writeIO(uint16_t regaddress, uint8_t data);
```

A terminal emulator such as putty or realterm will be used to convert your laptop into a dumb terminal. Your keyboard will then be the serial input device and your monitor will be used as the serial output device.