Andy Krock
ENP 341 – eLantern
5/20/15

File Structure

- **src**
  - Makefile
  - button_state.c/.h
  - io.h
  - light_state.c/.h
  - main.c
  - softuart.c/.h
  - timer.c/.h
- **doc**
  - Datasheet
  - DebounceStateMachine
  - ApplicationStateMachine
  - This Document(with code in .pdf form)
  - PowerPoint Presentation

Current functionality of my eLantern

- Selectable brightness – High, Medium, Low
- Fade Mode
- Runtime Shutoff
- Sleep mode
- SoftUart is partially implemented

Circuit Schematic/Pinouts

- Light = PB1
- Button = PB3
- RX = PB4
- TX = PB2

Code Structure

In my main loop the first thing that occurs is a variety of things are initialized. This includes my timer, interrupts, piezo, light and softuart. After these are initialized I run a main loop every 10ms. Currently three functions are being run in my main loop. The first is button_status which checks to see the buttin_flag flag has been set. If so the function does some debouncing. The next function that is called is the light_status function which sets the mode of the light depending on where you are in the state machine. The light_status contains states such as brightness level and fading. The last function in the loop currently does not work fully but in the future should print out to a serial port the current state of the system via the software uart.

MCU Hardware Description

- Timer 1
  - Timer 1 is used for all of my timing needs. Timer 1 is set up to trigger the interrupt every 1ms. This value was selected because 1ms is a nice easy number to work with. With a timer that triggers every 1ms I am able to easily say how long I want to run different commands.

- Interrupts

    - This interrupt is set so that any change to PCINT3, which is PB2, will result in an interrupt trigger. This is where my piezo is attached to the processor so when the piezo is tapped the light will do things accordingly.

- Timer 0
    - This is handled by the softuart.c file

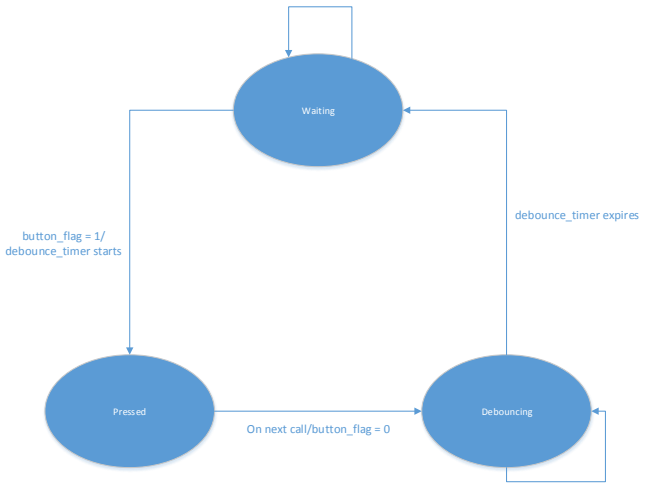
ISR Code

- Timer ISR
    - This timer ISR is executed every millisecond and adds a tick to all the timers.
- Interrupt ISR
    - This ISR is run whenever the state of PB2 changes. When the state changes the button_flag is set as long as the system is not debouncing.

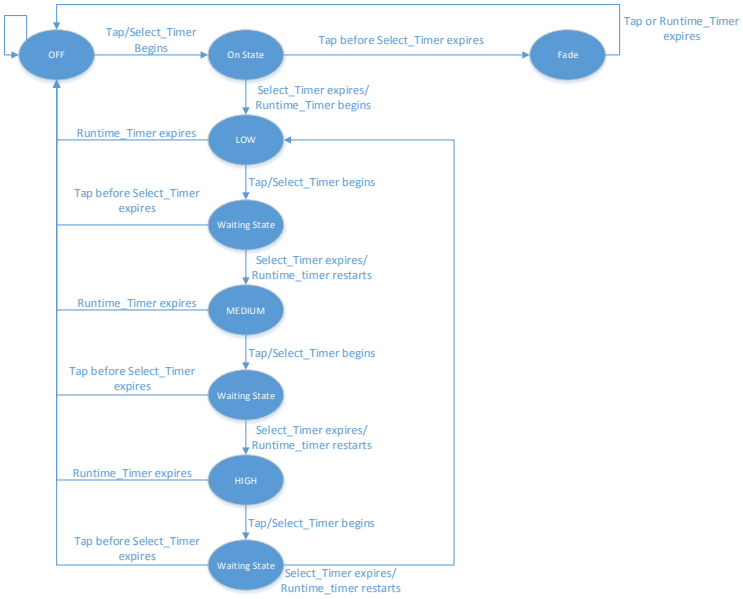
Software Modules

- Debounce
    - Once a tap is registered the state machince moves to PRESSED to reset the flag and then the state machine moves to DEBOUNCING. While in this state the system does not record any other presses. Once the debounce timer has expired the state machine moves back to WAITING.
- User
    - This one is more complex in nature. Every tap is only one tap, there is no two tap or many taps option. How this works is it depends how fast taps are tapped. To enter the section where you are able to select the brightness you tap only once. From there you can change the brightness by tapping once and then to turn off the device you tap two times in a row. To enter the fade state the device must first be off, and then you tap twice in a row.

Debounce State Machine



Application State Machine



Next Steps

- Include a module that counts how many times the piezo is tapped
- Make the exit of the fade state two taps instead of one
- Add the ability to change the speed of the fading
- Continual cleanup of code

```
/*-------------------------------------------------------------
* Author :          Andrew Krock
* Filename :        main.c
* Date Created :    Monday March 23, 2015 07:53:28 PM
* Last Edited :     Thursday May 14, 2015 11:12:14 PM
* Description :     This file handles the main job loop
                    that runs every eLanternServicePeriod
-------------------------------------------------------------*/

#define F_CPU 1000000

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

#include "timer.h"
#include "button_state.h"
#include "light_state.h"
#include "io.h"
#include "softuart.h"

unsigned int eLanternServicePeriod = SERVICE_PERIOD;
//unsigned int testPeriod = 1000;

//Description:
//This main loop runs every time get_ticks returns a value
//longer than eLanternServicePeriod

int main(void){
    timer_init();
    interrupt_init();
    button_init();
    light_init();
    softuart_init();
    while(1){
        if(get_ticks() > eLanternServicePeriod){    //Does these jobs every period d
efined
            eLanternServicePeriod += 10;    //Update the service period
            button_status();
            light_status();
            print_light_state();
        }
/*      if(get_ticks() > testPeriod){
            testPeriod += 1000;
            softuart_putchar('1');
        }*/
    }
    return 0;
}
```

```
/*-------------------------------------------------------
* Author :          Andrew Krock
* Filename :        io.h
* Date Created :    Monday March 23, 2015 08:01:43 PM
* Last Edited :     Friday May 15, 2015 04:51:10 PM
* Description :     This file handles all the macros of the
                    project
-------------------------------------------------------*/

#ifndef IO_H
#define IO_H

//Variables
#define SERVICE_PERIOD  10

//Light Pin
#define LIGHT_PIN       (1 << PORTB1)

//Buttion Pin
#define BUTTON_PIN      (1 << PORTB3)

//Button state Macros
#define WAITING         0
#define PRESSED         1
#define DEBOUNCING      2
#define DEBOUNCE_TIME   100 //in milliseconds

//Light State Macros
#define LIGHT_LOW       25
#define LIGHT_MEDIUM    75
#define LIGHT_HIGH      120
#define OFF             0
#define ON_STATE        1
#define LOW             2
#define WAITING_STATE_1 3
#define MEDIUM          4
#define WAITING_STATE_2 5
#define HIGH            6
#define WAITING_STATE_3 7
#define FADE            8
#define UP              1
#define DOWN            0
#define SLEEP_TIME      10000 //in milliseconds
#define SELECT_TIME     500
#define RUNTIME         20000

//Timer/Counter0 control register A
#define COM0A1x         (1<<COM0A1)
#define COM0A0x         (1<<COM0A0)
#define COM0B1x         (1<<COM0B1)
#define COM0B0x         (1<<COM0B0)
#define WGM01x          (1<<WGM01)
#define WGM00x          (1<<WGM00)

//Timer/Counter0 control register B
#define FOC0Ax          (1<<FOC0A)
#define FOC0Bx          (1<<FOC0B)
#define WGM02x          (1<<WGM02)
#define CS02x           (1<<CS02)
#define CS01x           (1<<CS01)
#define CS00x           (1<<CS00)

//Timer/Counter Interrupt Mask Register
#define OCIE0Ax         (1<<OCIE0A)
#define OCIE0Bx         (1<<OCIE0B)
#define TOIE1x          (1<<TOIE1)

//Timer/Counter Control1 Control Register
#define CTC1x           (1<<CTC1)
#define PWM1Ax          (1<<PWM1A)
#define COM1A1x         (1<<COM1A1)
#define COM1A0x         (1<<COM1A0)
#define CS13x           (1<<CS13)
```

```
#define CS12x           (1<<CS12)
#define CS11x           (1<<CS11)
#define CS10x           (1<<CS10)

//MCU Control Register
#define ISC00x          (1<<ISC00)
#define ISC01x          (1<<ISC01)

//GIMSK Genral Interrupt Mask Register
#define INT0x           (1<<INT0)
#define PCIEx           (1<<PCIE)

//PCMSK - Pin Change Mask Register/for interrupts
#define PCINT3x         (1<<PCINT3)

//MCUCR MCU Control Register
#define SM1x            (1<<SM1)
#define SEx             (1<<SE)

//SOFTUART Defs
#define SOFTUART_BAUD_RATE      2400

#define SOFTUART_RXPIN   PINB
#define SOFTUART_RXDDR   DDRB
#define SOFTUART_RXBIT   PB4

#define SOFTUART_TXPORT  PORTB
#define SOFTUART_TXDDR   DDRB
#define SOFTUART_TXBIT   PB2

#endif //IO_H
```

```
/*-------------------------------------------------------
* Author :          Andrew Krock
* Filename :        timer.h
* Date Created :    Monday March 23, 2015 07:59:34 PM
* Last Edited :     Saturday May 09, 2015 01:15:44 PM
* Description :
-------------------------------------------------------*/

#ifndef TIMER_H
#define TIMER_H

void timer_init();
unsigned int get_ticks();
unsigned int get_debounce();
unsigned int get_sleep();
unsigned int get_select();
unsigned int get_runtime();
unsigned int get_fade();

extern unsigned int ticks;
extern unsigned int debounce_timer;
extern unsigned int sleep_timer;
extern unsigned int select_timer;
extern unsigned int runtime_timer;
extern unsigned int fade_timer;

#endif //TIMER_H
```

```
/*-------------------------------------------------------------
* Author :          Andrew Krock
* Filename :        button_state.h
* Date Created :    Thursday March 26, 2015 01:34:01 PM
* Last Edited :     Thursday May 14, 2015 03:39:58 PM
* Description :
-------------------------------------------------------------*/

#ifndef BUTTON_STATE_H
#define BUTTON_STATE_H

void button_init();
void interrupt_init();
void button_status();

extern unsigned int button_flag;
#endif //BUTTON_STATE_H
```

```
/*-------------------------------------------------------
* Author :          Andrew Krock
* Filename :        light_state.h
* Date Created :    Thursday March 26, 2015 08:51:44 PM
* Last Edited :     Thursday May 14, 2015 03:40:00 PM
* Description :
-------------------------------------------------------*/

#ifndef LI_H
#define LI_H

void light_init();
void light_status();
void print_light_state();

#endif //LI_H
```

```
#if !defined(F_CPU)
//#warning F_CPU not defined in makefile - now defined in softuart.h
#define F_CPU 1000000UL
#endif

#include "io.h"

#define SOFTUART_T_COMP_LABEL      TIM0_COMPA_vect
#define SOFTUART_T_COMP_REG        OCR0A
#define SOFTUART_T_CONTR_REGA      TCCR0A
#define SOFTUART_T_CONTR_REGB      TCCR0B
#define SOFTUART_T_CNT_REG         TCNT0
#define SOFTUART_T_INTCTL_REG      TIMSK

#define SOFTUART_CMPINT_EN_MASK    (1<<OCIE0A)

#define SOFTUART_CTC_MASKA         (1<<WGM01)
#define SOFTUART_CTC_MASKB         (0)

/* "A timer interrupt must be set to interrupt at three times
   the required baud rate." */
#define SOFTUART_PRESCALE (8)
// #define SOFTUART_PRESCALE (1)

#if (SOFTUART_PRESCALE==8)
#define SOFTUART_PRESC_MASKA         (0)
#define SOFTUART_PRESC_MASKB         (1<<CS01)
#elif (SOFTUART_PRESCALE==1)
#define SOFTUART_PRESC_MASKA         (0)
#define SOFTUART_PRESC_MASKB         (1<<CS00)
#else
#error "prescale unsupported"
#endif


#define SOFTUART_TIMERTOP ( F_CPU/SOFTUART_PRESCALE/SOFTUART_BAUD_RATE/3 -1)

#if (SOFTUART_TIMERTOP > 0xff)
#warning "Check SOFTUART_TIMERTOP"
#endif

#define SOFTUART_IN_BUF_SIZE     32

// Init the Software Uart
void softuart_init(void);

// Clears the contents of the input buffer.
void softuart_flush_input_buffer( void );

// Tests whether an input character has been received.
unsigned char softuart_kbhit( void );

// Reads a character from the input buffer, waiting if necessary.
char softuart_getchar( void );

// To check if transmitter is busy
unsigned char softuart_can_transmit( void );

// Writes a character to the serial port.
void softuart_putchar( const char );

// Turns on the receive function.
void softuart_turn_rx_on( void );

// Turns off the receive function.
void softuart_turn_rx_off( void );

// Write a NULL-terminated string from RAM to the serial port
void softuart_puts( const char *s );

// Write a NULL-terminated string from program-space (flash)
// to the serial port. i.e. softuart_puts_p(PSTR("test"))
void softuart_puts_p( const char *prg_s );
```

```
// Helper-Macro - "automaticly" inserts PSTR
// when used: include avr/pgmspace.h before this include-file
#define softuart_puts_P(s___) softuart_puts_p(PSTR(s___))
```

```
/*-------------------------------------------------------
* Author :          Andrew Krock
* Filename :        timer.c
* Date Created :    Monday March 23, 2015 07:59:26 PM
* Last Edited :     Thursday May 14, 2015 02:47:28 PM
* Description :     This file handles timer setup,
                    counting and returning timer values
-------------------------------------------------------*/

#define F_CPU 1000000

#include <avr/sfr_defs.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "timer.h"
#include "io.h"

//Starting timers at 0
unsigned int ticks = 0;
unsigned int debounce_timer = 0;
unsigned int sleep_timer = 0;
unsigned int select_timer = 0;
unsigned int runtime_timer = 0;
unsigned int fade_timer = 0;

//Initializes timer to CTC for 1 ms period
void timer_init(){
    cli();
    //Start moving over to timer1
    TCCR1 = (CTC1x)|(PWM1Ax)|(COM1A1x)|(CS11x)|(CS10x);
    OCR1C = 250;
    TIMSK = (TOIE1x); //Overflow int
    OCR1A = 0;
    //To do PWM set OCR1A to a value
    sei();
}

//Function returns current tick value
unsigned int get_ticks(){
    unsigned int temp;
    cli();
    temp = ticks;
    sei();
    return temp;
}

//Function returns current debounce timer value
unsigned int get_debounce(){
    unsigned int temp;
    cli();
    temp = debounce_timer;
    sei();
    return temp;
}

//Function that returns sleep timer value
unsigned int get_sleep(){
    unsigned int temp;
    cli();
    temp = sleep_timer;
    sei();
    return temp;
}

//Function that returns select timer value
unsigned int get_select(){
    unsigned int temp;
    cli();
    temp = select_timer;
    sei();
    return temp;
}
```

```
//Function that returns runtime timer value
unsigned int get_runtime(){
    unsigned int temp;
    cli();
    temp = runtime_timer;
    sei();
    return temp;
}

//Function that returns the fade timer value
unsigned int get_fade(){
    unsigned int temp;
    cli();
    temp = fade_timer;
    sei();
    return temp;
}

//Interrupts every 1 ms and adds a tick
ISR(TIMER1_OVF_vect){
    ticks ++;
    debounce_timer ++;
    sleep_timer ++;
    select_timer ++;
    runtime_timer ++;
    fade_timer ++;
}
```

```c
/*-------------------------------------------------------
* Author :          Andrew Krock
* Filename :        button_state.c
* Date Created :    Thursday March 26, 2015 01:21:11 PM
* Last Edited :     Thursday May 14, 2015 03:40:08 PM
* Description :     This file handles the button state
-------------------------------------------------------*/

#define F_CPU 1000000

#include <avr/sfr_defs.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "button_state.h"
#include "io.h"
#include "timer.h"

unsigned int button_flag = 0;
unsigned int button_state = WAITING;

//Inits the button
void button_init(){
    PORTB |= BUTTON_PIN;
}

//Initializes the interrupt vector
void interrupt_init(){
    cli();
    GIMSK = (PCIEx);
    PCMSK = (PCINT3x);
    sei();
}


//Debounce switch statement
void button_status(){
    switch(button_state){
        case WAITING:
            if(button_flag == 1){
                debounce_timer = 0;
                button_state = PRESSED;
            }
            break;
        case PRESSED:
            button_flag = 0;
            button_state = DEBOUNCING;
            break;
        case DEBOUNCING:
            if(get_debounce() > DEBOUNCE_TIME){
                button_state = WAITING;
            }
            break;
        default:
            break;
    }
}

ISR(PCINT0_vect){
    if(button_state == DEBOUNCING){
        button_flag = 0;
    }
    else{
        button_flag = 1;
    }
}
```

```
/*-------------------------------------------------------
* Author :          Andrew Krock
* Filename :        light_state.c
* Date Created :    Thursday March 26, 2015 08:51:28 PM
* Last Edited :     Friday May 15, 2015 04:51:08 PM
* Description :     This function handles the state of the
                    light
-------------------------------------------------------*/
 #define F_CPU 1000000

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

#include "light_state.h"
#include "button_state.h"
#include "timer.h"
#include "io.h"
#include "softuart.h"

unsigned int temp = 100;
char char_light_state;

//Inits light state
unsigned int light_state = OFF;

//Inits fade direction
unsigned int direction = UP;

//Inits state of light pin
void light_init(){
    DDRB |= LIGHT_PIN;
}

void print_light_state(){
    if(light_state != temp){
        char_light_state = (char)light_state;
        softuart_putchar(char_light_state);
        //softuart_putchar('0');
        //temp = light_state;
    }
    temp = light_state;
}

//Finite state machine that controls what the
//light is doing
void light_status(){
    switch(light_state){
        case OFF:
            OCR1A = OFF;
            if(button_flag == 1){
                light_state = ON_STATE;
                select_timer = 0;
            }
            //if(get_sleep() > SLEEP_TIME){
            //  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
            //  sleep_mode();
            //}
            break;
        case ON_STATE:
            if(button_flag == 1 && get_select() < SELECT_TIME){
                OCR1A = OFF;
                fade_timer = 0;
                runtime_timer = 0;
                light_state = FADE;
            }
            else if(get_select() > SELECT_TIME){
                light_state = LOW;
                runtime_timer = 0;
            }
            break;
        case LOW:
```

```
            OCR1A = LIGHT_LOW;
            if(button_flag == 1){
                light_state = WAITING_STATE_1;
                select_timer = 0;
            }
            if(get_runtime() > RUNTIME){
                light_state = OFF;
            }
            break;
        case WAITING_STATE_1:
            if(button_flag == 1 && get_select() < SELECT_TIME){
                light_state = OFF;
            }
            else if(get_select() > SELECT_TIME){
                runtime_timer = 0;
                light_state = MEDIUM;
            }
            break;
        case MEDIUM:
            OCR1A = LIGHT_MEDIUM;
            if(button_flag == 1){
                light_state = WAITING_STATE_2;
                select_timer = 0;
            }
            if(get_runtime() > RUNTIME){
                light_state = OFF;
            }
            break;
        case WAITING_STATE_2:
            if(button_flag == 1 && get_select() < SELECT_TIME){
                light_state = OFF;
            }
            else if(get_select() > SELECT_TIME){
                runtime_timer = 0;
                light_state = HIGH;
            }
            break;
        case HIGH:
            OCR1A = LIGHT_HIGH;
            if(button_flag == 1){
                light_state = WAITING_STATE_3;
                select_timer = 0;
            }
            if(get_runtime() > RUNTIME){
                light_state = OFF;
            }
            break;
        case WAITING_STATE_3:
            if(button_flag == 1 && get_select() < SELECT_TIME){
                light_state = OFF;
            }
            else if(get_select() > SELECT_TIME){
                runtime_timer = 0;
                light_state = LOW;
            }
            break;
        case FADE:
            if(get_fade() > 50){
                if(direction == UP && OCR0B < 121){
                    OCR1A++;
                    if(OCR1A == 120){
                        direction = DOWN;
                    }
                }
                if(direction == DOWN && OCR0B > 0){
                    OCR1A--;
                    if(OCR1A == 1){
                    direction = UP;
                    }
                }
                fade_timer = 0;
            }
            if(button_flag == 1 || get_runtime() > RUNTIME){
```

```
                light_state = OFF;
            }
            break;
        default:
            break;
    }
}
```

```
// softuart.c
// AVR-port of the generic software uart written in C
//
// Generic code from
// Colin Gittins, Software Engineer, Halliburton Energy Services
// (available from the iar.com web-site -> application notes)
//
// Adapted to AVR using avr-gcc and avr-libc
// by Martin Thomas, Kaiserslautern, Germany
// <eversmith@heizung-thomas.de>
// http://www.siwawi.arubi.uni-kl.de/avr_projects
//
// AVR-port Version 0.3  4/2007
//
// ----------------------------------------------------------------------
//
// Remarks from Colin Gittins:
//
// Generic software uart written in C, requiring a timer set to 3 times
// the baud rate, and two software read/write pins for the receive and
// transmit functions.
//
// * Received characters are buffered
// * putchar(), getchar(), kbhit() and flush_input_buffer() are available
// * There is a facility for background processing while waiting for input
// The baud rate can be configured by changing the BAUD_RATE macro as
// follows:
//
// #define BAUD_RATE  19200.0
//
// The function init_uart() must be called before any comms can take place
//
// Interface routines required:
// 1. get_rx_pin_status()
//    Returns 0 or 1 dependent on whether the receive pin is high or low.
// 2. set_tx_pin_high()
//    Sets the transmit pin to the high state.
// 3. set_tx_pin_low()
//    Sets the transmit pin to the low state.
// 4. idle()
//    Background functions to execute while waiting for input.
// 5. timer_set( BAUD_RATE )
//    Sets the timer to 3 times the baud rate.
// 6. set_timer_interrupt( timer_isr )
//    Enables the timer interrupt.
//
// Functions provided:
// 1. void flush_input_buffer( void )
//    Clears the contents of the input buffer.
// 2. char kbhit( void )
//    Tests whether an input character has been received.
// 3. char getchar( void )
//    Reads a character from the input buffer, waiting if necessary.
// 4. void turn_rx_on( void )
//    Turns on the receive function.
// 5. void turn_rx_off( void )
//    Turns off the receive function.
// 6. void putchar( char )
//    Writes a character to the serial port.
//
// ----------------------------------------------------------------------

/*
Remarks by Martin Thomas (avr-gcc):
V0.1:
- stdio.h not used
- AVR-Timer in CTC-Mode ("manual" reload may not be accurate enough)
  Timer1 used here (Timer0 CTC not available i.e. on ATmega8)
- Global Interrupt Flag has to be enabled (see Demo-Application)
- Interface timer_set and set_timer_interrupt not used here
- internal_tx_buffer was defined as unsigned char - thas could not
  work since more than 8 bits needed, changed to unsigned short
- some variables moved from "global scope" into ISR function-scope
```

```
- GPIO initialisation included
- Added functions for string-output inspired by P. Fleury's AVR UART-lib.
V0.2:
- adjust num of RX-bits
- adapted to avr-libc ISR-macro (replaces SIGNAL)
- disable interrupts during timer-init
- used unsigned char (uint8_t) where apropriate
- removed "magic" char checking (0xc2)
- added softuart_can_transmit()
- Makefile based on template from WinAVR 1/2007
- reformated
- extended demo-application to show various methods to
  send a string from flash and RAM
- demonstrate usage of avr-libc's stdio in demo-applcation
- tested with ATmega644 @ 3,6864MHz system-clock using
  avr-gcc 4.1.1/avr-libc 1.4.5 (WinAVR 1/2007)
V0.3
- better configuration options in softuart.h.
  ->should be easier to adapt to different AVRs
- tested with ATtiny85 @ 1MHz (int R/C) with 2400 bps
- AVR-Studio Project-File
*/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

#include "softuart.h"

#define SU_TRUE 1
#define SU_FALSE 0

// startbit and stopbit parsed internaly (see ISR)
#define RX_NUM_OF_BITS (8)
volatile static char             inbuf[SOFTUART_IN_BUF_SIZE];
volatile static unsigned char    qin  = 0;
/*volatile*/ static unsigned char qout = 0;
volatile static unsigned char    flag_rx_off;
volatile static unsigned char    flag_rx_ready;

// 1 Startbit, 8 Databits, 1 Stopbit = 10 Bits/Frame
#define TX_NUM_OF_BITS (10)
volatile static unsigned char  flag_tx_ready;
volatile static unsigned char  timer_tx_ctr;
volatile static unsigned char  bits_left_in_tx;
volatile static unsigned short internal_tx_buffer; /* ! mt: was type uchar - this wa
s wrong */

#define set_tx_pin_high()      ( SOFTUART_TXPORT |=  ( 1<<SOFTUART_TXBIT ) )
#define set_tx_pin_low()       ( SOFTUART_TXPORT &= ~( 1<<SOFTUART_TXBIT ) )
#define get_rx_pin_status()    ( SOFTUART_RXPIN  & ( 1<<SOFTUART_RXBIT ) )
// #define get_rx_pin_status() ( ( SOFTUART_RXPIN & ( 1<<SOFTUART_RXBIT ) ) ? 1 : 0
)

ISR(SOFTUART_T_COMP_LABEL)
{
    static unsigned char flag_rx_waiting_for_stop_bit = SU_FALSE;
    static unsigned char rx_mask;

    static char timer_rx_ctr;
    static char bits_left_in_rx;
    static unsigned char internal_rx_buffer;

    char start_bit, flag_in;
    char tmp;

    // Transmitter Section
    if ( flag_tx_ready ) {
        tmp = timer_tx_ctr;
        if ( --tmp <= 0 ) { // if ( --timer_tx_ctr <= 0 )
            if ( internal_tx_buffer & 0x01 ) {
                set_tx_pin_high();
            }
```

```
            else {
                set_tx_pin_low();
            }
            internal_tx_buffer >>= 1;
            tmp = 3; // timer_tx_ctr = 3;
            if ( --bits_left_in_tx <= 0 ) {
                flag_tx_ready = SU_FALSE;
            }
        }
        timer_tx_ctr = tmp;
    }

    // Receiver Section
    if ( flag_rx_off == SU_FALSE ) {
        if ( flag_rx_waiting_for_stop_bit ) {
            if ( --timer_rx_ctr <= 0 ) {
                flag_rx_waiting_for_stop_bit = SU_FALSE;
                flag_rx_ready = SU_FALSE;
                inbuf[qin] = internal_rx_buffer;
                if ( ++qin >= SOFTUART_IN_BUF_SIZE ) {
                    // overflow - rst inbuf-index
                    qin = 0;
                }
            }
        }
        else {  // rx_test_busy
            if ( flag_rx_ready == SU_FALSE ) {
                start_bit = get_rx_pin_status();
                // test for start bit
                if ( start_bit == 0 ) {
                    flag_rx_ready      = SU_TRUE;
                    internal_rx_buffer = 0;
                    timer_rx_ctr       = 4;
                    bits_left_in_rx    = RX_NUM_OF_BITS;
                    rx_mask            = 1;
                }
            }
            else {  // rx_busy
                if ( --timer_rx_ctr <= 0 ) {
                    // rcv
                    timer_rx_ctr = 3;
                    flag_in = get_rx_pin_status();
                    if ( flag_in ) {
                        internal_rx_buffer |= rx_mask;
                    }
                    rx_mask <<= 1;
                    if ( --bits_left_in_rx <= 0 ) {
                        flag_rx_waiting_for_stop_bit = SU_TRUE;
                    }
                }
            }
        }
    }
}

static void avr_io_init(void)
{
    // TX-Pin as output
    SOFTUART_TXDDR |=  ( 1 << SOFTUART_TXBIT );
    // RX-Pin as input
    SOFTUART_RXDDR &= ~( 1 << SOFTUART_RXBIT );
}

static void avr_timer_init(void)
{
    unsigned char sreg_tmp;

    sreg_tmp = SREG;
    cli();

    SOFTUART_T_COMP_REG = SOFTUART_TIMERTOP;     /* set top */

    SOFTUART_T_CONTR_REGA = SOFTUART_CTC_MASKA | SOFTUART_PRESC_MASKA;
```

```
    SOFTUART_T_CONTR_REGB = SOFTUART_CTC_MASKB | SOFTUART_PRESC_MASKB;

    SOFTUART_T_INTCTL_REG |= SOFTUART_CMPINT_EN_MASK;

    SOFTUART_T_CNT_REG = 0; /* reset counter */

    SREG = sreg_tmp;
}

void softuart_init( void )
{
    flag_tx_ready = SU_FALSE;
    flag_rx_ready = SU_FALSE;
    flag_rx_off   = SU_FALSE;

    set_tx_pin_high(); /* mt: set to high to avoid garbage on init */
    avr_io_init();

    // timer_set( BAUD_RATE );
    // set_timer_interrupt( timer_isr );
    avr_timer_init(); // replaces the two calls above
}

static void idle(void)
{
    // timeout handling goes here
    // - but there is a "softuart_kbhit" in this code...
    // add watchdog-reset here if needed
}

void softuart_turn_rx_on( void )
{
    flag_rx_off = SU_FALSE;
}

void softuart_turn_rx_off( void )
{
    flag_rx_off = SU_TRUE;
}

char softuart_getchar( void )
{
    char ch;

    while ( qout == qin ) {
        idle();
    }
    ch = inbuf[qout];
    if ( ++qout >= SOFTUART_IN_BUF_SIZE ) {
        qout = 0;
    }

    return( ch );
}

unsigned char softuart_kbhit( void )
{
    return( qin != qout );
}

void softuart_flush_input_buffer( void )
{
    qin  = 0;
    qout = 0;
}

unsigned char softuart_can_transmit( void )
{
    return ( flag_tx_ready );
}

void softuart_putchar( const char ch )
{
```

```
    while ( flag_tx_ready ) {
        ; // wait for transmitter ready
          // add watchdog-reset here if needed;
    }

    // invoke_UART_transmit
    timer_tx_ctr       = 3;
    bits_left_in_tx    = TX_NUM_OF_BITS;
    internal_tx_buffer = ( ch<<1 ) | 0x200;
    flag_tx_ready      = SU_TRUE;
}

void softuart_puts( const char *s )
{
    while ( *s ) {
        softuart_putchar( *s++ );
    }
}

void softuart_puts_p( const char *prg_s )
{
    char c;

    while ( ( c = pgm_read_byte( prg_s++ ) ) ) {
        softuart_putchar(c);
    }
}
```