Lund University
Faculty of Engineering

Digital project

EITF40

All Year Automatic Illuminator

Authors:
Niklas Aldén
Max Andersson

Supervisor:
Bertil Lindvall

March 18, 2016

Abstract

This project tackles the problem of illuminating an area in an efficient way during the dark hours of the day. At sunset, the light is turned on until people go to sleep. During the night, the light is turned off to preserve power. However, if something were to move in the area during night, the light should come on for a couple of minutes. In the morning, before sunrise, the light turns on in case someone wakes up early. The proposed solution is a prototype that uses an ATmega16 together with a real-time clock, a motion sensor, and a light sensor to turn on an LED at the correct time of the day.

Contents

1 Introduction

A farm or a smaller village can sometimes be experienced as a dark and unfriendly environment. Without public streetlights, ones property will not be illuminated after sunset. With our product we eliminate the need to venture out into the dark in the morning as well as coming home to a dark house in the evening. It will also provide the feeling of security in the middle of the night with a motion sensor that turn on the light when someone approaches your home.

1.1 Report Outline

Section 2 lists the specifications for the project.

Section 3 contains a description of the major components along with the software used to program the microcontroller.

Section 4 describes how the implementation was done.

Section 5 describes the finished product.

Section 6 contains the discussion of the project and the problems that were encountered.

Appendix A gives a step-by-step user guide on how to operate and set up the clock and timers.

Appendix B contains the source code and header file for the AVR microcontroller.

2 Specification

The following characteristics and features was needed:

- Turn on LED when sun sets
- Turn off LED when clock reach midnight ± chosen offset
- Turn on LED when clock reach dawn minus chosen offset
- Turn off LED when the sun has risen
- Activate motion sensor during the night
- Deactivate motion sensor during day
- Activate light sensor during day
- Deactivate light sensor during night
- Settings should be possible to access via a user interface

After doing settings once, the system should be fully automatic as long as power is delivered to the circuit.

3 Components and Software

3.1 ATmega16

The ATmega16 is an 8-bit AVR RISC-based microcontroller with 16KB of programmable flash memory, 1KB SRAM, 512B EEPROM, an 8-channel 10-bit A/D converter, and a JTAG interface for on-chip debugging. The device supports a throughput up to 16MIPS at 16MHz[1] and operates with a supply voltage between 4.5V – 5.5V. Using the internal crystal, 8MHz is the maximum clock-frequency. However, since the design is not time-critical, the frequency is set to just 2MHz, resulting in lower power consumption. The lower clock-rate also makes the internal timers/counters increment slower, allowing for more convenient usage [1].

[1] 16MHz with an external crystal

3.2 Real-Time Clock/Calender

The Real-Time Clock/Calender (RTCC) tracks the time and date using an external 32.768kHz crystal [2]. It tracks time using several internal registers and automatically adjusts for months with fewer than 31 days, including corrections for leap years. The RTCC has two settable alarms, both connected to one I/O pin, triggering on second, minute, hour, day of the week, date, or month. For communication, the RTCC uses the I^2C bus. Together with a backup battery or low-leakage capacitance, the RTCC can keep track of the time and date even during a power outage. To communicate with the RTCC over the I^2C protocol, a reference design from [3] was used.

3.3 Motion Sensor

To detect motion, a Passive Infrared (PIR) sensor was used. When connected to a supply voltage and ground, the PIR sensor notifies that a motion has been detected by setting the output pin high. When no motion is detected, the output pin is set low [4].

3.4 Light Sensor

The light sensor used in this project operates in the visible light spectrum and has a linear sensitivity. It consists of a phototransistor with a light-sensitive base where the photons in the light generate the base–collector current of an NPN-transistor. When light shines on the prototransistor, the channel between the collector and emitter will conduct. With no light, the collector-emitter channel is cut-off [5].

3.5 Liquid Crystal Display

To display information to the user, a Liquid Crystal Display (LCD) is used. It consists of two rows with sixteen characters each. The characters has a resolution of 7×5 pixels. To interface the LCD, a reference design found at [6] was used.

3.6 Atmel Studio

To program the AVR microcontroller, Atmel Studio 6.2 was used. The computer and AVR was connected using a JTAG interface. All software is written in C.

4 Implementation

Implementation was done in several steps, implementing the different components one at a time. This resulted in having a skeleton of code for each component that later could be combined into a complete design. This decreased the design time and increased the possibility of debugging.

To start with, supply voltage and ground was routed to the AVR. As this was done, testing of the in -and output pins could be tested simply by connecting an LED to one of the pins. Because of the various use of interrupts in the design, the next thing to implement would be interrupt handling routines. By routing a push button to one of the external interrupt pins, together with an internal pull-up resistor, the interrupt could easily be tested. This was done by blinking the LED whilst the corresponding interrupt routine was executed on the AVR.

By having a working interrupt routine when pushing a button, a settings menu for the LCD could be constructed. When pushing the button, the menu change state. When the menu had transversed through all states it returned to the default state. A watchdog timer was also implemented so that the user would be sent back to the default state after 10 seconds of idle usage. This was implemented using the AVR's internal 16-bit counter. When the counter overflows, it generates an interrupt which was used to reset the state of the menu. With the LCD menu implemented, the LCD could be used to print values of registers or state of counters, etc. which proved useful during development of the rest of the design.

Next step in the design was to include the light sensor (phototransistor). To create a convenient way to adjust the threshold on the fly, the AVR's internal analog comparator was used. The comparator takes two input pins, one connected to an adjustable reference voltage and one connected to the

input from the phototransistor. The comparator generates a logic '1' when the input voltage from the phototransistor is larger than the reference voltage, otherwise it generates a logic '0'. This can be interpret as light or dark. This was also tested using the LED, switching the LED on when it is dark.

The motion sensor should turn on the LED during nighttime and the LED should stay lit for a short period of time after no motion has been detected. To accomplish this, the motion sensor was connected to an external interrupt pin. When the interrupt was triggered, the controller cleared one of the internal 8-bit counters and turned on the LED. When the interrupt pin goes low, the controller waits for 76 counter overflows, correlating to approximately 10 seconds, as shown in (1), before tuning off the LED. If a new interrupt occurred during this delay time, the counter resets its value.

$$OVF_{\text{counter}} = \frac{t_{\text{delay}} \cdot f_{\text{CPU}}}{2^{\text{bits}} \cdot N_{\text{prescaler}}} = \frac{10\,\text{s} \cdot 2\text{MHz}}{2^8 \cdot 1024} \approx 76, \tag{1}$$

where:

t_{delay} = the length of the wanted delay,
f_{CPU} = clock frequency of AVR,
bits = number of bits for counter,
$N_{\text{prescaler}}$ = size of counter prescaler.

The I^2C communication protocol [3] uses a two-wire-interface (TWI) to connect the AVR, as a master, to one or more slave devices. This communication is done using a shared communication line for both receiving and transmitting data. The RTCC was connected to the two TWI pins on the AVR. These consist of one serial clock (SCL) -and one serial data line (SDA). SCL is used to synchronize the data transfers whilst all data is sent over the SDA. When implemented, the first test was to write data to the RTCC's internal registers and then validate completion using read-back values. When the connection was working properly, the alarms were programmed and tested by connecting the alarm output pin on the RTCC to an external interrupt pin on the AVR.

5 Results

The design is working as expected and a picture of it is shown in Figure 1. To be able to adapt to the seasons, one need to know when the sun rises. This is done by sampling the clock each morning at dawn. The sampled time is then used the next day, together with the chosen offset, to determine when to turn on the LED. The first day one cannot know when the sun rises, so to relieve the user of the discomfort of looking this up, the controller needs one day of calibration. The first day, sunrise is set to 6 AM as default.

The design has proven resilient to power outage, keeping its settings during a whole weekend, without a power supply plugged in. At this time the backup capacitor had dropped from 4.6V to 3.0V.

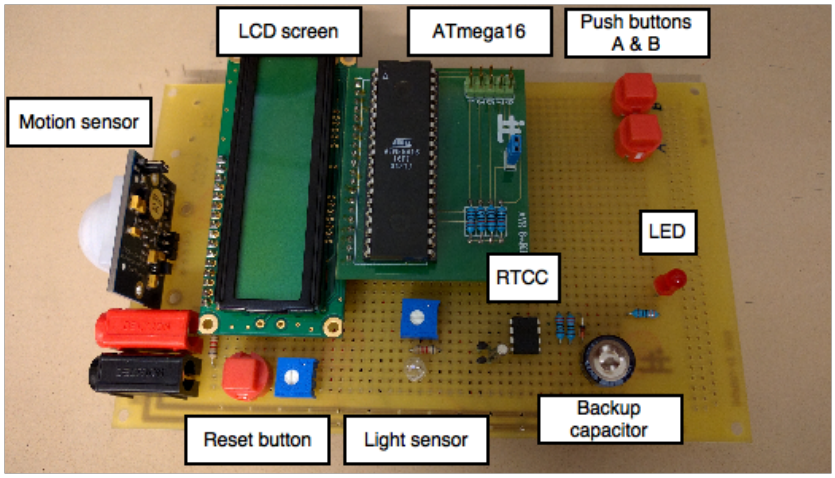

Figure 1: Final product

The AVR's Flash -and SRAM storage was not fully utilized, as shown in Table 1. The number of pins used on the AVR sums up to 24 out of 40 possible[2], as shown in Figure 2. Something that is worth noticing is that all external interrupt pins are used, further discussed in Section 6.1.

[2]24 pins including 4 pins for the JTAG interface. Figure 2 shows 20 connected pins.

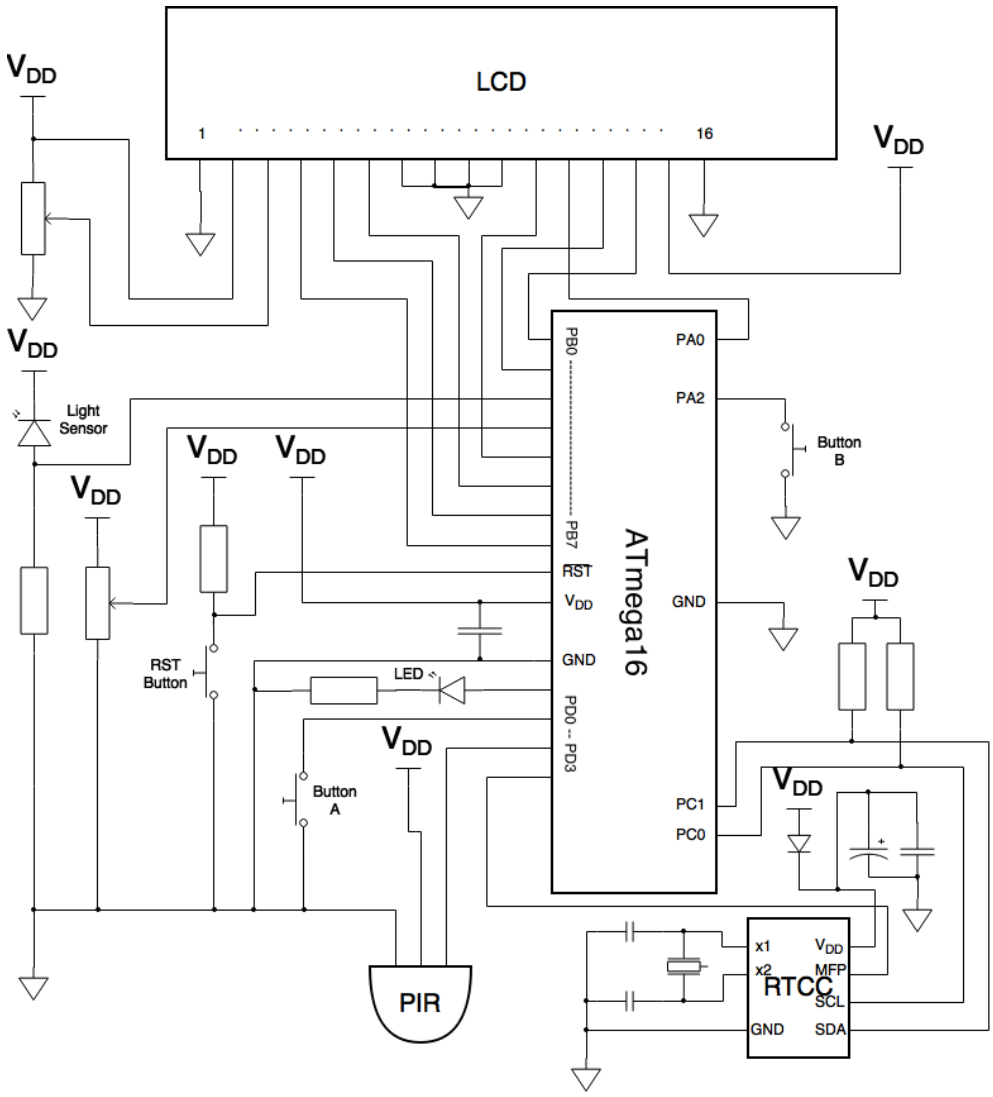

Figure 2: Final schematic

Table 1: Memory usage

Memory	**bytes**	%
Program memory	5604/16K	34
Data memory	205/1K	20

6 Discussion

This design is very versatile. It could be used for almost anything that relies on performing cyclic behavior. In this implementation we use a motion sensor and a phototransistor to determine when to light up an LED, but it could also be used to run a pump at a given time, or make a cup of coffee in the morning. With that said, we believe that the content of this report is good knowledge for people interested in home automation and technical solutions in general.

6.1 Problems

The goal was to use interrupts, both internal and external, for as many things as possible in the design. Interrupts allows for a very effective execution of the code, since no time is wasted checking if a bit is set in a register or not, i.e. polling. Unfortunately, there are only three external interrupt pins on the ATmega16 and one of these pins doubles as one of the inputs to the analog comparator. This meant we had to prioritize which peripherals should be connected to the external interrupts. In the end, we opted for polling the two buttons whilst connecting the alarm-pin, from the RTCC, and the output from the motion sensor to the the two remaining external interrupt pins. If the microcontroller had more external interrupt pins, the execution of the program would have more effective, with a lower power dissipation as a bonus.

The RTCC has several alarm-trigger-setting. However, there is no option for triggering on a match on hour, minute, and second without regarding the date. To avoid unnecessary complex settings, we chose to trigger only on hour-matches. Otherwise, the AVR would need to keep track of the date as well, and reprogram the date of the alarm every single day. This resulted in less flexibility for the alarm-time. On the other hand, the design is less error-prompt.

References

[1] ATmega16, http://www.atmel.com/devices/ATMEGA16.aspx, 12:00 2016-02-11

[2] Datasheet Real-Time Clock MCP7940M-I/P, https://www.elfa.se/Web/Downloads/_t/ds/MCP7940M_eng_tds.pdf?mime=application%2Fpdf, 12:20 2016-02-11

[3] Manual and code for I^2C communication, http://homepage.hispeed.ch/peterfleury/avr-software.html, 10:05 2016-02-23

[4] Datasheet Passive Infrared Sensor, https://www.parallax.com/sites/default/files/downloads/555-28027-PIR-Sensor-Prodcut-Doc-v2.2.pdf, 09:35 2016-02-23

[5] Datasheet Light Sensor, https://www.elfa.se/Web/Downloads/ta/_e/xoAMS3_Data_E.pdf?mime=application%2Fpdf, 09:45 2016-02-23

[6] Manual and code for LCD, http://maxembedded.com/2011/06/lcd-interfacing-with-avr/, 10:00 2016-02-23

A User Guide

By preforming the following steps, you will setup the starting conditions of your very own *All Year Automatic Illuminator*. First, after plugging in the power, you will be able to adjust the time of the day if not correct.

Note: Idle usage will return to default menu after approximately 10 seconds.

Start-up

This should be done each time the circuit has been out of power more then a few days.

1. Press **A** to enter the setup menu.
2. Press **B** to enter clock setup menu.
3. Press **B** to increment time (seconds, minutes, hours).
   Press **A** to set targeted value.
4. After setting hours, confirm time setting by pressing **A** once more.
5. Press **RST** button to initialize the circuit in right state of the day. After one day, calibration of the morning alarm is automatically done.

Setup morning timer

This can be used to set the offset. Alarm will set to $t_{\text{sunrise}} - t_{\text{offset}}$. For example, if the offset is 1 hour and the sun rises at 7 AM, the LED will turn on at 6 AM and turn off when the light sensor detects the sunrise (7 AM).

1. Press **A** to enter setup menu.
2. Press **A** to skip clock setup.
3. Press **B** to enter morning timer setup.
4. Press **B** to set offset (0 – 4 hours)
   Press **A** to confirm.

Setup midnight timer

The LED turns off at midnight, plus or minus an offset, 12 PM $\pm\, t_{\text{offset}}$.

1. Press **A** to enter setup menu.
2. Press **A** to skip clock setup.
3. Press **A** to skip morning timer setup.
4. Press **B** to enter midnight timer setup.
5. Press **B** to set offset ($00{:}00 \pm 2$ hours)
   Press **A** to confirm.

View current timer settings

Pressing **A** four times, skipping the setup menus, will display the current times for the morning -and midnight timer.

B Source Code

Listing 1: illuminator.h

```
#ifndef BLINK_H
#define BLINK_H

#define F_CPU                (2000000UL) // 2 MHz clock speed
#define MOTION_DELAY         (76)
#define LIGHT_DELAY          (38)

#define RTC_READ             (0xDF)
#define RTC_WRITE            (0xDE)
#define ALARM_DAWN_ADR       (0x0A)
#define ALARM_MIDNIGHT_ADR   (0x11)

#include <stdio.h>
#include <string.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "lcd.h"
#include "i2c_master.h"

typedef struct {
    uint8_t sec;
    uint8_t min;
    uint8_t hour;
} time;

enum Light_sensor
```

```
{
    light,
    dark
};


// GLOBAL variables
extern uint8_t state, substate;
extern int dawn_offset, dusk_offset;
extern uint8_t timer0_ovf_cnt;

void init();
void rst();
void lcd();
int BCD2int(uint8_t);
uint8_t int2BCD(int);
void print_time(time);
void set_clock(time);
time get_time();
void set_alarm(time, uint8_t);
time get_alarm(uint8_t);
void disp_menu(time);

#endif
```

Listing 2: illuminator.c

```
#include "blink.h"

time alarm_dawn_time;
time alarm_midnight_time;
uint8_t state, substate;
int dawn_offset, midnight_offset;
uint8_t timer0_ovf_cnt, timer2_ovf_cnt;
uint8_t light_ctrl_reg;

void init()
{
    alarm_dawn_time.hour = 6;
    alarm_dawn_time.min = 0;
    alarm_dawn_time.sec = 0;
    alarm_midnight_time.hour = 0;
    alarm_midnight_time.min = 0;
    alarm_midnight_time.sec = 0;
    time curr_time = get_time();    // Get current time
    state = substate = 0;
    dawn_offset = midnight_offset = 0;
    light_ctrl_reg = 0;
    set_alarm(alarm_dawn_time, ALARM_DAWN_ADR);
    set_alarm(alarm_midnight_time, ALARM_MIDNIGHT_ADR);
    timer0_ovf_cnt = timer2_ovf_cnt = 0;

    i2c_start(RTC_WRITE);
    i2c_write(0x07);                                    // write to ctrl
   register
    i2c_write((BCD2int(curr_time.hour) < 6) ? 0x10 : 0x20);         // turn on alarm0,
   turn off alarm1
```

```
 i2c_stop();

 if (BCD2int(curr_time.hour) < 6) {                                    // 0 < time <
morning alarm
     ACSR |= 1 << ACD;         // turn off analog comparator
     PORTD &= ~(1 << PD0);     // Turn off LED
     GICR |= 1 << INT0;        // enable INT0 (PIR)
 } else if (!(ACSR & (1 << ACO)) && BCD2int(curr_time.hour) < 12) {  // morning
alarm < time < 12
     PORTD |= (1 << PD0);      // Turn on LED
     GICR &= ~(1 << INT0);     // disable INT0 (PIR)
 } else if (ACSR & (1 << ACO)) {                                       // 12 < time <
evening
     PORTD &= ~(1 << PD0);     // Turn off LED
     GICR &= ~(1 << INT0);     // disable INT0 (PIR)
 } else {                                                              // evening <
time < midnight alarm
     ACSR |= 1 << ACD;         // turn off analog comparator
     PORTD |= (1 << PD0);      // Turn on LED
     GICR &= ~(1 << INT0);     // disable INT0 (PIR)
 }

 // LED
 DDRD |= (1 << PD0);       // PD0 (LED) as output

 // Light sensor
 DDRB &= ~((1 << PB2) | (1 << PB3)); // AIN0, AIN1 as inputs
 PORTB |= ((1 << PB2) | (1 << PB3)); // enable pull up resistors

 // PIR interrupt pin
 DDRD &= ~(1 << PD2);      // PD2 as input
```

```
    // RTC interrupt pin
    DDRD &= ~(1 << PD3);     // PD3 as input
    PORTD |= 1 << PD3;       // enable pull up resistor

    // Button A
    DDRD &= ~(1 << PD1);     // PD1 as input
    PORTD |= 1 << PD1;       // enable pull up resistor

    // Button B
    DDRA &= ~(1 << PA2);     // PA2 as input
    PORTA |= 1 << PA2;       // enable pull up resistor

    // Interrupts
    GICR |= 1 << INT1;           // enable INT1 (RTC) always

    MCUCR |= ((1 << ISC00) | (1 << ISC01) | (1 << ISC10) | (1 << ISC11));    // trigger
   INT0 and INT1 on rising edge
    sei();       // Set External Interrupt

    // Timer / Counter
    TIMSK |= ((1 << TOIE0) | (1 << TOIE2)); // Enable overflow interrupt for counter 0
   and 2
    TCCR1B |= (1 << CS12); // Start and set prescaler for internal 16 bit counter 1


}

void rst()
{
    // blink 5 times to notify reset
    for (int i = 0; i < 10; ++i) {
        PORTD ^= 1 << PD0;
```

```
            _delay_ms(42);
        }
    }

    void lcd()
    {
        lcd_init(LCD_DISP_ON); //display (cursor) style
    }

    int BCD2int(uint8_t bcd)
    {
        int i = (bcd & 0x70) >> 4;
        return i * 10 + (int)(bcd & 0x0F);
    }

    uint8_t int2BCD(int i)
    {
        return ((i / 10) << 4) | (i % 10);
    }

    void print_time(time t)
    {
        char str[17] = {0};      // clear string for LCD-row
        char buf[5] = {0};       // clear string for time
        if (!(t.hour & 0x70)) strcat(str,"0");  // add leading 0 if < 10
        sprintf(buf,"%d:", BCD2int(t.hour));
        strcat(str,buf);
        if (!(t.min & 0x70)) strcat(str,"0");   // add leading 0 if < 10
        sprintf(buf,"%d:", BCD2int(t.min));
        strcat(str,buf);
        if (!(t.sec & 0x70)) strcat(str,"0");   // add leading 0 if < 10
        sprintf(buf,"%d", BCD2int(t.sec));
```

```
    strcat(str,buf);
    lcd_puts(str);
}

void set_clock(time t)
{
    i2c_start(RTC_WRITE);
    i2c_write(0x00);
    i2c_write(t.sec | (1 << 7) );
    i2c_write(t.min);
    i2c_write(t.hour);
    i2c_stop();
}

time get_time()
{
    time curr_time;
    i2c_start(RTC_WRITE);
    i2c_write(0x00);
    i2c_start(RTC_READ);
    curr_time.sec = i2c_read_ack();
    curr_time.min = i2c_read_ack();
    curr_time.hour = i2c_read_nack();
    i2c_stop();
    return curr_time;
}

void set_alarm(time alarm_time, uint8_t address)
{
    if(address == ALARM_DAWN_ADR){
        alarm_time.hour = int2BCD(BCD2int(alarm_time.hour) - dawn_offset);
    } else {
```

```
        alarm_time.hour = (midnight_offset < 0) ? int2BCD(24 + midnight_offset) :
    int2BCD(midnight_offset);
    }

    i2c_start(RTC_WRITE);
    i2c_write(address);
    i2c_write(alarm_time.sec);
    i2c_write(alarm_time.min);
    i2c_write(alarm_time.hour);
    i2c_write(0xA0);    // Set alarm to trigger on hour
    i2c_stop();
}

time get_alarm(uint8_t address)
{
    time curr_alarm;
    i2c_start(RTC_WRITE);
    i2c_write(address);
    i2c_start(RTC_READ);
    curr_alarm.sec = i2c_read_ack();
    curr_alarm.min = i2c_read_ack();
    curr_alarm.hour = i2c_read_nack();
    i2c_stop();
    return curr_alarm;
}

void disp_menu(time new_time)
{
    time curr_time;
    lcd_clrscr();           // clear screen of lcd
    lcd_home();             // bring cursor to 0,0
    char str[17] = {0};     // clear string for LCD-row
```

```
char buf[5] = {0};       // clear string for time

switch (state) {
    case 0:
        curr_time = get_time();
        lcd_puts("Press for setup!");
        lcd_gotoxy(0,1);
        print_time(curr_time);
        _delay_ms(50);
        break;

    case 1:
        switch (substate) {
            case 0:
                curr_time = get_time();
                lcd_puts("Set clock? A = N");
                lcd_gotoxy(0,1);
                print_time(curr_time);
                lcd_gotoxy(11,1);
                lcd_puts("B = Y");
                _delay_ms(50);
                break;
            case 1:
                print_time(new_time);
                lcd_gotoxy(0,1);
                lcd_puts("sec: ");
                if (!(new_time.sec & 0x70)) strcat(str,"0");     // add leading 0 if
< 10
                sprintf(buf,"%d",BCD2int(new_time.sec));         // int-to-string
                strcat(str,buf);                                 // concatenate time
                lcd_puts(str);
                break;
```

```
                case 2:
                    print_time(new_time);
                    lcd_gotoxy(0,1);
                    lcd_puts("min: ");
                    if (!(new_time.min & 0x70)) strcat(str,"0");     // add leading 0 if
< 10
                    sprintf(buf,"%d",BCD2int(new_time.min));        // int-to-string
                    strcat(str,buf);                                // concatenate time
                    lcd_puts(str);
                    break;
                case 3:
                    print_time(new_time);
                    lcd_gotoxy(0,1);
                    lcd_puts("hour: ");
                    if (!(new_time.hour & 0x70)) strcat(str,"0");    // add leading 0 if
< 10
                    sprintf(buf,"%d",BCD2int(new_time.hour));       // int-to-string
                    strcat(str,buf);                                // concatenate time
                    lcd_puts(str);
                    break;
                default:
                    lcd_puts("error 1");
                    break;
                }
            break;
        case 2:
            switch(substate) {
                case 0:
                    lcd_puts("Set dawn timer?");
                    lcd_gotoxy(0,1);
                    lcd_puts("A = N, B = Y");
                    break;
```

```
            case 1:
                lcd_puts("Timer set to:");
                lcd_gotoxy(0,1);
                lcd_puts("Dawn - ");
                sprintf(buf,"%d h",(int)dawn_offset);    // int-to-string
                strcat(str,buf);                         // concatenate time
                lcd_puts(str);
                break;
            default:
                lcd_puts("error 2");
                break;
            }
        break;
    case 3:
        switch(substate) {
            case 0:
                lcd_puts("Set 00:00 timer?");
                lcd_gotoxy(0,1);
                lcd_puts("A = N, B = Y");
                break;
            case 1:
                lcd_puts("Timer set to:");
                lcd_gotoxy(0,1);
                lcd_puts("Midnight +- ");
                sprintf(buf,"%d h",(int)midnight_offset);    // int-to-string
                strcat(str,buf);                             // concatenate time
                lcd_puts(str);
                break;
            default:
                lcd_puts("error 3");
                break;
            }
```

```
            break;

        case 4:
            lcd_puts("Dawn: ");
            print_time(get_alarm(ALARM_DAWN_ADR));
            lcd_gotoxy(0,1);
            lcd_puts("Night: ");
            print_time(get_alarm(ALARM_MIDNIGHT_ADR));
            break;

        default:
            lcd_puts("WOW");
            break;
    }
    _delay_ms(50);
}

// Watchdog timer, if no button were pressed
ISR(TIMER1_OVF_vect)
{
    state = 0;                          // go to default menu
    dawn_offset = midnight_offset = 0;  // clear offset settings if not finished
}

// Interrupt from PIR sensor
ISR(INT0_vect)
{
    TCCR0 |= ((1 << CS02) | (1 << CS00));   // Start and set prescaler for internal 8
   bit counter 0
    TCNT0 = 0;                              // Clear counter 0
    PORTD |= 1 << PD0;                      // Turn on LED
    timer0_ovf_cnt = 0;                     // Clear overflow counter for counter 0
```

```
}

ISR(TIMER0_OVF_vect)
{
    timer0_ovf_cnt = (~PIND & (1 << PD2)) ? timer0_ovf_cnt + 1 : timer0_ovf_cnt;    //
    only increment if pin is low, i.e. no motion detected
}

// Interrupt from RTC alarm
ISR(INT1_vect)
{
    if (!(GICR & (1 << INT0))) {    // PIR interrupt is off = before midnight
        ACSR |= 1 << ACD;           // turn off analog comparator
        GICR |= 1 << INT0;          // turn on interrupt from PIR sensor
        PORTD &= ~(1 << PD0);       // turn off LED
        // turn off alarm0, turn on alarm1
        i2c_start(RTC_WRITE);
        i2c_write(0x07);            // write to ctrl register
        i2c_write(0x10);            // turn off alarmMidnight, turn on alarmDawn
        i2c_stop();
        light_ctrl_reg = 0;         // enable sampling of dawn time
    } else {                        // PIR interrupt is on = between midnight and dawn
        ACSR &= ~(1 << ACD);        // turn on analog comparator
        GICR &= ~(1 << INT0);       // turn off interrupt from PIR sensor
        PORTD |= 1 << PD0;          // turn on LED
        // turn on alarm0, turn off alarm1
        i2c_start(RTC_WRITE);
        i2c_write(0x07);            // write to ctrl register
        i2c_write(0x20);            // turn on alarmDawn, turn off alarmMidnight
        i2c_stop();
    }
```

```
    i2c_start(RTC_WRITE);
    i2c_write(0x0D);    // Alarm 0
    i2c_write(0xA0);    // Set alarm to trigger on hour
    i2c_stop();
    i2c_start(RTC_WRITE);
    i2c_write(0x14);    // Alarm 1
    i2c_write(0xA0);    // Set alarm to trigger on hour
    i2c_stop();
}

ISR(TIMER2_OVF_vect)
{
    ++timer2_ovf_cnt;
    if (timer2_ovf_cnt >= LIGHT_DELAY) {
        if (!(PORTD & (1 << PD0))) {    // LED is off before timer overflows
            // update dawn alarm
            alarm_dawn_time = get_time();
            set_alarm(alarm_dawn_time, ALARM_DAWN_ADR);
            light_ctrl_reg = 1;
        }

        timer2_ovf_cnt = 0;
        TCCR2 &= ~((1 << CS22) | (1 << CS21) | (1 << CS20));    // turn off counter 2
    }
}



//  Main function

int main(void)
{
```

```
init();
rst();
lcd();
i2c_init();


time new_time = {0};

while (1) {
    disp_menu(new_time); //apply changes menu

    // Button A pressed?
    if((~PIND & (1 << PD1))){
        _delay_ms(20);                  // debouncer
        while ((~PIND & (1 << PD1)));   // wait for button to be release
        _delay_ms(50);                  // debouncer
        TCNT1H = 0x00;                  // clear counter
        TCNT1L = 0x00;

        switch (state) {
            case 0:             // just display current time
                state = 1;
                substate = 0;
                break;

            case 1:             // set new clock time?
                switch (substate) {
                    case 0:
                        state = 2;
                        substate = 0;
                        break;
```

```
            case 1:      // set seconds
                state = 1;
                substate = 2;
                break;

            case 2:      // set minutes
                state = 1;
                substate = 3;
                break;

            case 3:      // set hours
                state = 2;
                substate = 0;
                set_clock(new_time);
                break;

            default:
                break;
        }
        break;

    case 2:              // set dawn offset?
        switch (substate) {
            case 0:
                state = 3;
                substate = 0;
                break;

            case 1:      // set dawn offset
                state = 3;
                substate = 0;
                set_alarm(alarm_dawn_time, ALARM_DAWN_ADR);
```

```
                break;

            default:
                break;
        }
        break;

    case 3:             // set midnight offset?
        switch (substate) {
            case 0:
                state = 4;
                substate = 0;
                break;

            case 1:     // set midnight offset
                state = 4;
                substate = 0;
                set_alarm(alarm_midnight_time, ALARM_MIDNIGHT_ADR);
                break;

            default:
                break;
        }
        break;

    case 4:          // display alarms
        state = 0;
        substate = 0;
        break;

    default:
        break;
```

```
    }

    if (state == 0) {
        TIMSK &= ~(1 << TOIE1);      // disable overflow interrupt
    } else {
        TIMSK |= (1 << TOIE1);       // enable overflow interrupt
    }
}

// Button B pressed?
if ((~PINA & (1 << PA2))) {
    _delay_ms(20);                   // debouncer
    while ((~PINA & (1 << PA2)));    // wait for button to be release
    _delay_ms(50);                   // debouncer
    TCNT1H = 0x00;                   // clear Counter
    TCNT1L = 0x00;

    switch (state) {
        case 0:              // just display current time
            // Button B, no effect
            break;

        case 1:              // set new clock time?
            switch (substate) {
                case 0:
                    state = 1;
                    substate = 1;
                    new_time = get_time();
                    break;

                case 1:      // set seconds
                    new_time.sec = int2BCD(BCD2int(new_time.sec) + 1);
```

```
                    if(BCD2int(new_time.sec) > 59) new_time.sec = 0;
                    break;

                case 2:      // set minutes
                    new_time.min = int2BCD(BCD2int(new_time.min) + 1);
                    if(BCD2int(new_time.min) > 59) new_time.min = 0;
                    break;

                case 3:      // set hours
                    new_time.hour = int2BCD(BCD2int(new_time.hour) + 1);
                    if(BCD2int(new_time.hour) > 23) new_time.hour = 0;
                    break;

                default:
                    break;
            }
            break;

        case 2:              // set dawn offset?
            switch (substate) {
                case 0:
                    state = 2;
                    substate = 1;
                    break;

                case 1:      // set dawn offset
                    dawn_offset = (dawn_offset > 3) ? 0 : dawn_offset + 1;
                    break;

                default:
                    break;
            }
```

```
                    break;

                case 3:              // set midnight offset?
                    switch (substate) {
                        case 0:
                            state = 3;
                            substate = 1;
                            break;

                        case 1:      // set midnight offset
                            midnight_offset = (midnight_offset > 1) ? -2 :
midnight_offset + 1;
                            break;

                        default:
                            break;
                    }
                    break;

                case 4:
                    // Button B, no effect
                    break;

                default:
                break;
            }
        }



        // Mask out analog compare bit
        if (ACSR & (1 << ACO) && !(ACSR & (1 << ACD))) {
```

```
            if (PIND & (1 << PD0) && !light_ctrl_reg) {
                TCCR2 |= ((1 << CS22) | (1 << CS21) | (1 << CS20)); // start counter 2
    with prescaler 1024
            }
            PORTD &= ~(1 << PD0);
        } else if (!(ACSR & (1 << ACD))) {
            PORTD |= 1 << PD0;
        }

        // Turn off LED if no motion has been detected for a while and pin is low
        if ((timer0_ovf_cnt >= MOTION_DELAY) && (~PIND & (1 << PD2))) {
            TCCR0 &= ~((1 << CS02) | (1 << CS00));  //  Stop internal 8 bit counter 0
            PORTD &= ~(1 << PD0);
            timer0_ovf_cnt = 0;
        }
    }
}
```