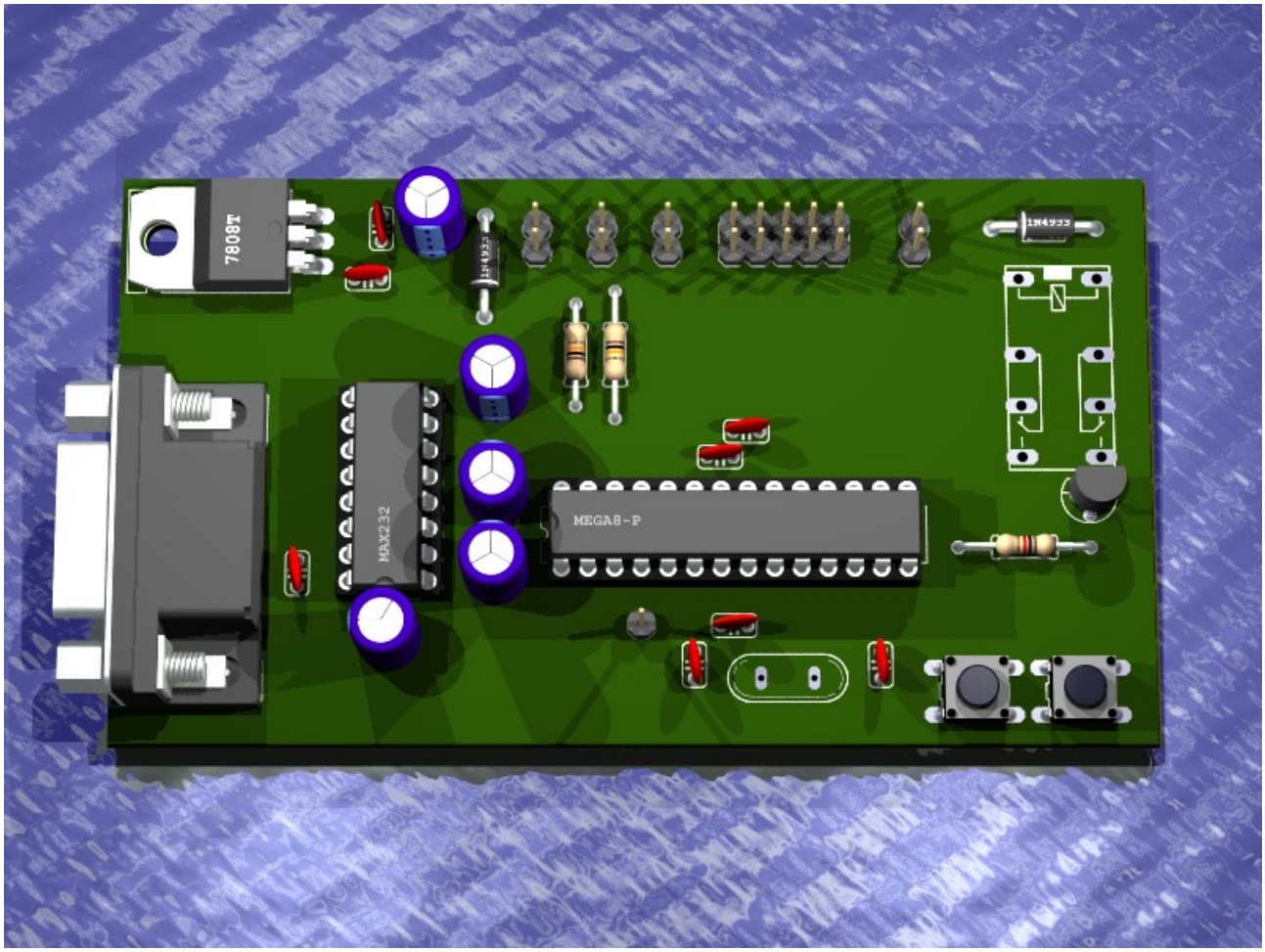
Amiv Bastli Uhr II

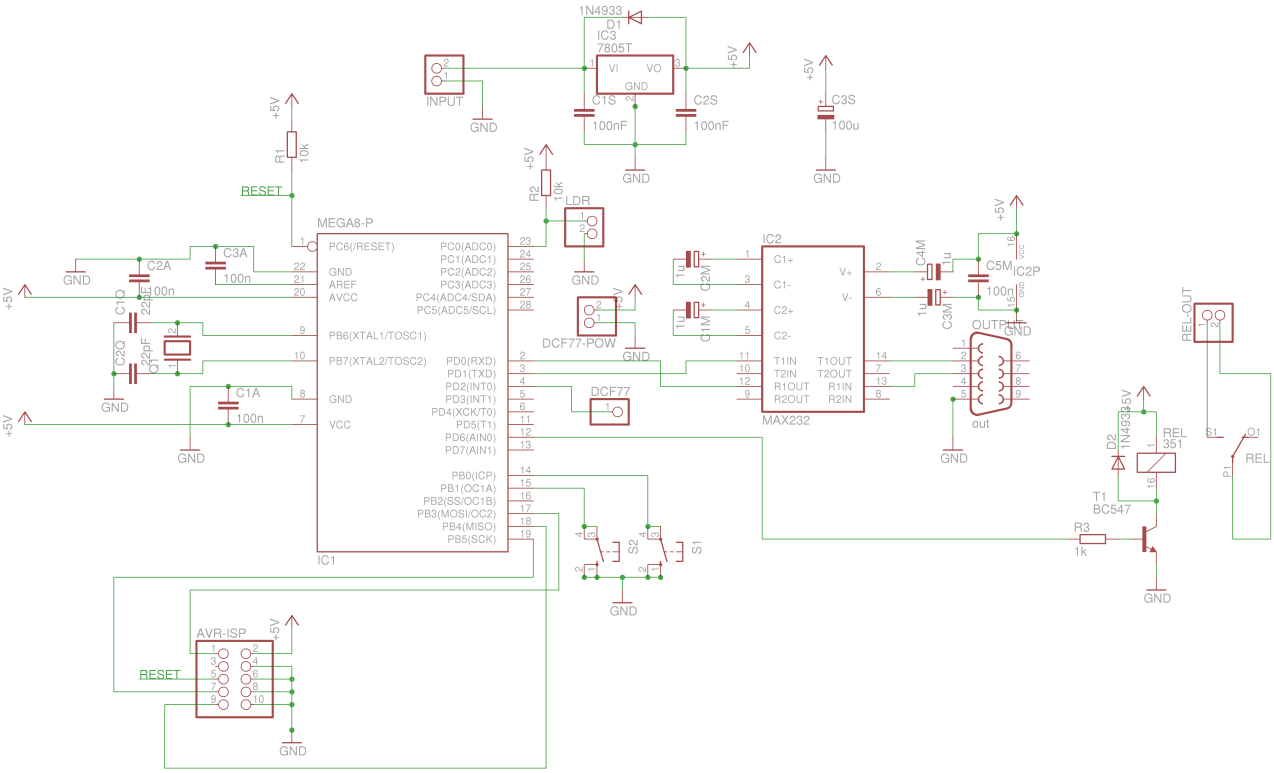
Fabio Banfi & Jonas Pfaff



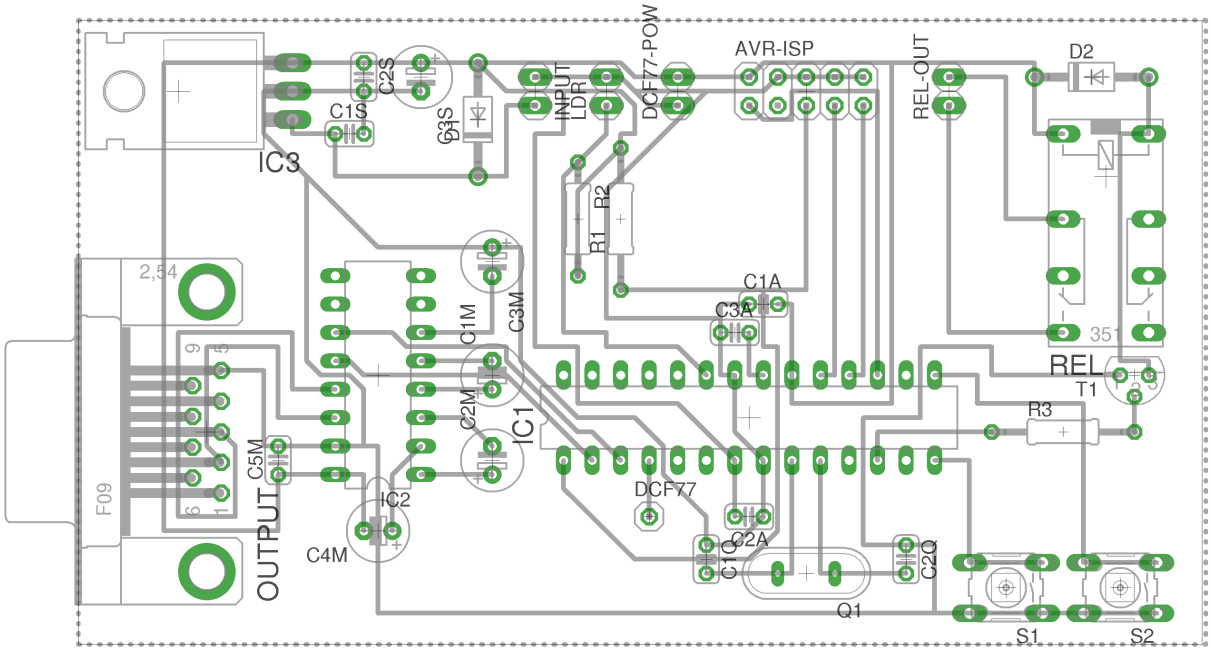
Index

Hardware

Schaltbild



Layout



Komponenten

Mikrocontroller
ATMEGA88-P

I/O

INPUT:	12V DC
REL-OUT:	Monitor VCC
LDR:	Foto Resistence
DCF77-POW:	DCF77 module power
DCF77:	DCF77 INPUT
AVR-ISP:	AVR Programmer
OUT:	RS232 OUTPUT monitor

Kondensatoren

C1A:	100 nF
C2A:	100 nF
C3A:	100 nF
C1M:	1 µF Elko
C2M:	1 µF Elko
C3M:	1 µF Elko
C4M:	1 µF Elko
C5M:	100 nF
C1Q:	22 pF
C2Q:	22 pF
C1S:	100 nF
C2S:	100 nF
C3S:	100 µF Elko

Dioden

D1:	1N4148
D2:	1N4148

IC's

IC1:	MEGA88-P
IC2:	MAX232
IC3:	7805T

Oszillator

Q1:	4096000Hz

Widerstände

R1:	10k Ω
R2:	10k Ω
R3:	1k Ω

Relais

REL:	351 Relais

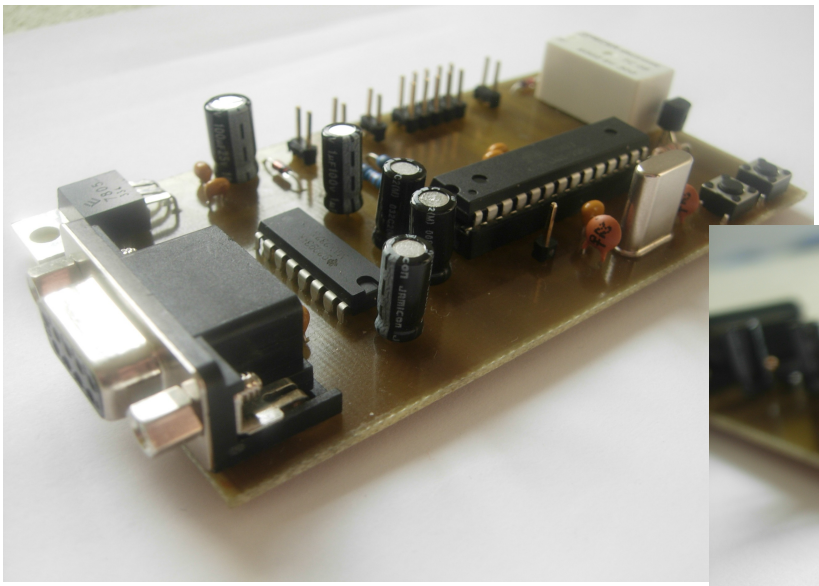
Schalter:

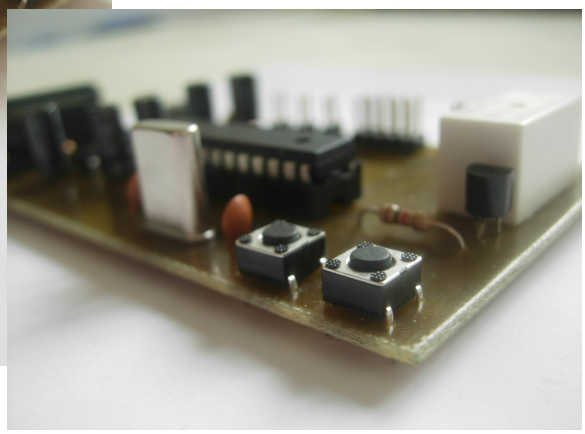
S1:	Printschalter
S2:	Printschalter

Transistor

T1:	BC546

Lichtsensor
Photoresistor





DCF77 – Funkuhrempfänger

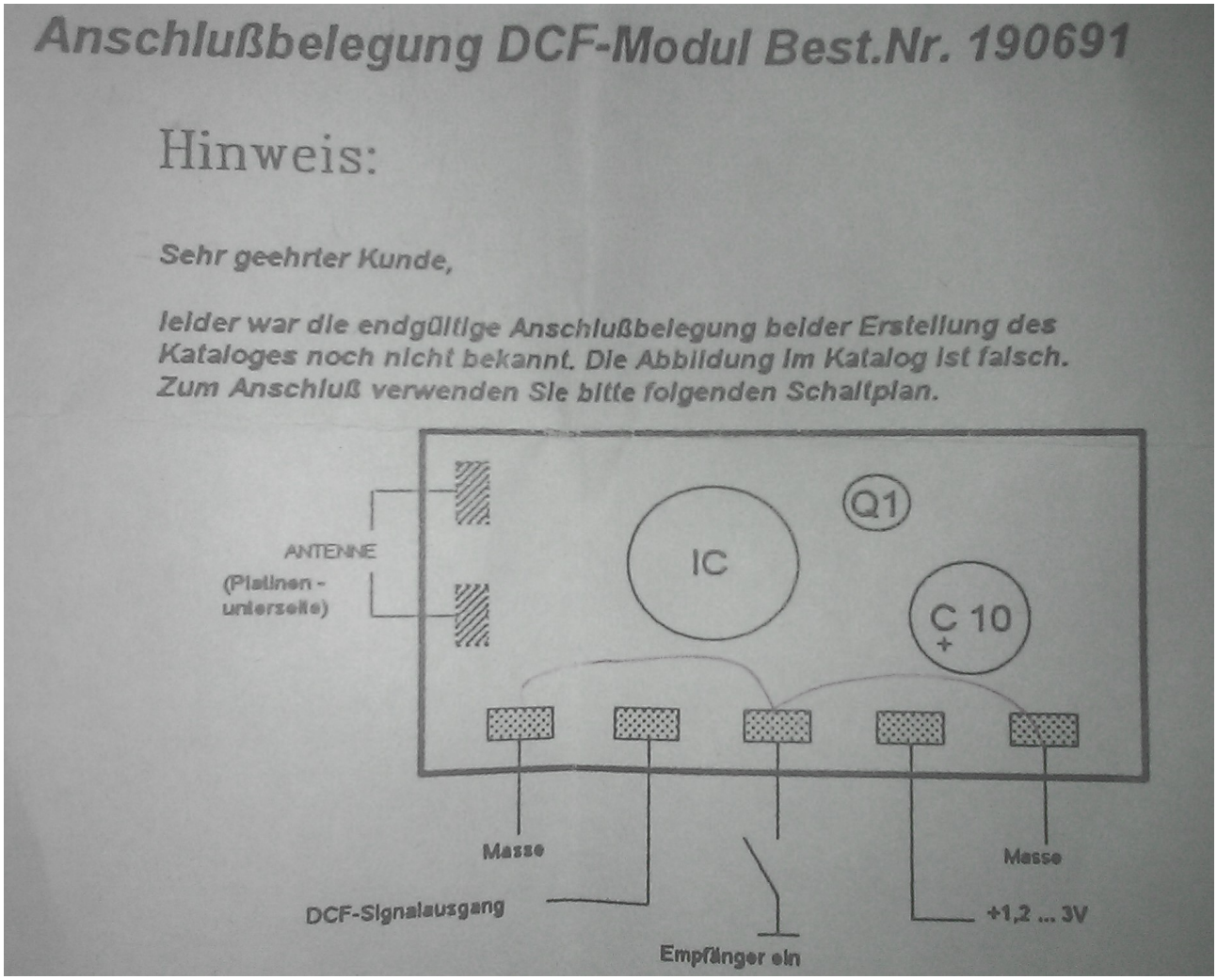
Signal

Sekunde	Bedeutung
0.	Minutenbeginn --> immer '0'
1.-14.	keine Bedeutung, normal '0' **NEU Wetterdaten(verschlüsselt)**
15.	Reserveantenne aktiv
16.	Umstellung zwischen Sommer- bzw. Winterzeit
17.	Sommerzeit
18.	Winterzeit
19.	Schaltsekunde
20.	Zeitbeginn
21.-27.	Minute 1, 2, 4, 8, 10, 20, 40
28.	Prüfbit Minute
29.-34.	Stunde 1, 2, 4, 8, 10, 20
35.	Prüfbit Stunde
36.-41.	Tag 1, 2, 4, 8, 10, 20
42.-44.	Wochentag 1, 2, 4
45.-49.	Monat 1, 2, 4, 8, 10
50.-57.	Jahr 1, 2, 4, 8, 10, 20, 40, 80
58.	Prüfbit Datum (36. - 57.)
59.	wird nicht gesendet (siehe oben)

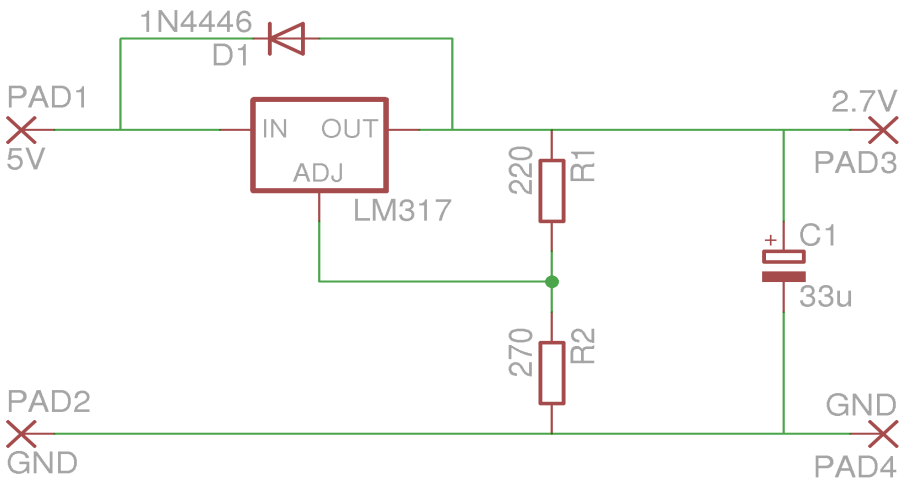
Signal Beispiel (10ms polling (3+...)x100 pol = 3+... s):

11111111111111111111111000000000011111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
|STEP ONE im Fall, dass hier (irgendwo) angefangen wird, wird 75ms gewartet | bis hier etwa.
00000000011111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111000000000000000000000011111
|Signal Anfang (1 mal pro sekunde), falls dauer 10 ms =~ 10 abtastungen Logisch 0. dauer 20ms Logisch 1.
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
Kein signal in dieser Sekunde = letzte Sekunde der Minute.|
111111111111111111111111111111111111111111111111000000000000000000001111111111111111111111111111111111111111111111111111111111111111111
|Beginn neuer Minute immer mit 20ms Signal = logisch 1.

Anschlussbelegung



Versorgung (5V → 2.7V)



Software

Main (*bawatch.c*)

Erklärung

Die Main enthält die Initialitation der diversen I/O Resourcen:

- 2 Taster (A + B)
- Monitor mit UART über MAX232
- DCF77-Antenne
- 16 bit Timer
- AD Converter für Lichtsensor

Ausserdem werden die wichtigsten globalen Variablen definiert.
Als erstes wird dann die Uhrzeit mit der DCF77-Antenne Empfangen.
Es folgt die Anschaltung des Monitors über das Relais, und eine Begrüssung. In der Endloschleife wird dann, falls die Zeit geupdated werden muss, die Funktion um die aktuelle Zeit auszugeben aufgerufen, der Lichtsensor wird kontrolliert (Monitor an und aus) und die Tasten Ausgabe um in das Menu zu kommen.

Funktionen & Variablen

```
int volatile second,time,update; //Timer und Zeitvariablen
int format;      //Falls 0 24h format, falls 1 12h format

int main (void)
    int light;
    int oldlight;
```

DCF77 (*dcf.c*)

Erklärung

Das *dcf.c* File enthält zwei Funktionen auch die direkt auf globale Variablen zugreifen.

Das return gestaltet sich dann als Boolean, 1 falls die Funktion erfolgreich abgschlossen wurde oder eben 0 falls nicht.

Die Funktion *dcffunction()*, tastet das Signal aus der Antenne und dem DCF-Modul alle 10ms ab. Bei der ersten while-Schleife geht es darum nicht in der mitte eines Signals anfangen auszuwerten. In der zweiten hingegen wird die Anzahl einsen und nullen, die vom Anfang eines Signals bis zum Anfang des nächsten Signals auftreten. Sobald eine Sekunde vorbei ist (Anfang neues Signal) werden die Zahlen ausgewertet und eine entsprechende Logisch “1” oder “0” in das Array gespreichert.

Die Form des Signals wurde oben dokumentiert.

Sobald eine Minute vorbei ist, und die ganze Minute abgetastet wurde (über eine Sekunde lang kein Signal) wird die zweite Funktion aufgerufen: *evalbits(int array[])* bekommt als Argument das gespeicherte array mit den Binärdaten die ausgewertet und in eine reale Zeit, mir berücksichtigung des Parity bits, umgewandelt werden.

Funktionen & Variablen

```
int dcffunction(void);
        int pointer=0;
        int ones=0;
        int zeros;
        int dcfarray[59];
        int bit;
        int temp;
        int errors;

int evalbits(int array[]);
        int minutes;
        int hours;
        int parity;
```

UART (*puts.c*)

Erklärung

Das File *puts.c* enthält alle Variablen und Funktionen die dem Output auf dem Monitor dienen. Die einzige wichtige Eigenschaft, die nicht in diesem File enthalten ist, ist die Baudrate, die im *bawatch.c* berechnet wird.
Die Variablen mit den ASCII-Arts die am Ende auf dem Bildschirm zu sehen sind werden zusammen mit dem Programm im Flash gespeichert, da es sonst im RAM eng wird. Ausserdem sind die Zahlen Kodiert (komprimiert) damit nicht so viel Platz gebraucht wird.

- putc sendet einen Charakter
- puts sendet einen ganzen String
- puthour werden dazu gebraucht die aktuelle Uhrzeit auszugeben
- putsomething ist als generelle Funktion gedacht, die aus dem Flash liest und ausgibt

Funktionen & Variablen

```
static unsigned char PROGMEM hello[];    //Begrüssungstext.
static unsigned char PROGMEM AM[];
static unsigned char PROGMEM PM[];
static unsigned char PROGMEM digits[10][5 * 16];

int uart_putc(unsigned char);

void uart_puts(char*);

void uart_puthour(int, int, int, int);

void puthour(int, int, int, int);

void uart_putsomething(char*);
        int symbols
```

Lichtsensor (*light.c*)

Erklärung

Dieses File beinhaltet nur eine Funktion, die die Lichtintensität auswertet und als Zahl zurückgibt.

Funktionen & Variablen

int readlight(void);

Menu (*menu.c*)

Erklärung

Diese Funktionen sind für die ausgabe des Menus verantwortlich. Im menu kann das format (12h – 24h) oder die Uhrzeit eingestellt werden.
Kombinationen:

Uhrzeit:	A+B	→	Main Menu
Main Menu:	A	→	Set hour
	B	→	Set format
Set hour:	A	→	Increase minutes/hour
	B	→	Change hour → Change minutes → Return
Set Format:	A	→	24h Format
	B	→	12h Format
	A+B	→	Return

Funktionen & Variablen

static unsigned char PROGMEM menu_title[881];
static unsigned char PROGMEM menu_main[1121];
static unsigned char PROGMEM menu_set_hour[241];
static unsigned char PROGMEM menu_set_format[1121];

void menu(void);

void set_format(void);

void set_hour(void);

Interrupts

16 bit Timer Interrupt

Der 16 bit Timer Interrupt wird jedes Mal wenn der Timer auf 16000 gezählt hat ausgelöst. Dies entspricht genau 4 Sekunden. Dabei wird der *TIMER1_COMPA_vect* geschaltet und die *ISR* Funktion erhöht die Variable *second*. Sobald 15 mal 4 Sekunden verstrichen sind wird die Zeitvariable *time* aktualisiert und ein Update des Outputs gefordert.

Anhang

Programme

- *bawatch.c*
- *puts.c*
- *dcf.c*
- *light.c*
- *menu.c*

Layout

- Eagle Schematic (digital)
- Eagle Board (digital)
- 1:1 Layout

Datasheets

- Atmega88 datasheet (digital)

Diverses

- Fotos / Bilder (digital)

```c
/** bawatch.c
  * BAWATCH v2.9.1
  *
  * Jonas Pfaff & Fabio Banfi 2011
  */

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

#define F_CPU 4096000UL
#define BAUDRATE 9600UL
#define UBRR_VAL ((F_CPU)/(BAUDRATE*16)-1)

int volatile second,time,update; //Timer und Zeitvariablen
int format=0;
#include "menu.h"
#include "puts.h"
#include <util/delay.h>
#include "menu.c"
#include "puts.c"
#include "dcf.c"
#include "light.c"

int main (void) {

/*Initialitation START ----------------------------------------------*/

    //Taster Init
    PORTB |= ((1 << DDB0) | (1 << DDB1)); // Pull-Up wiederstand ON
    DDRB &= ~((1 << DDB0) | (1 << DDB1)); // B0, B1, Eingänge.

    //Init Uart:
    UCSR0B = (1 << RXEN0) | (1 << TXEN0);
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00 | (1 << USBS0));
    UBRR0H = (UBRR_VAL >> 8);
    UBRR0L = UBRR_VAL;

    //Timer1 16bit
    TCCR1A = 0x00;
    TCCR1B |= ((1 << WGM12) | (1 << CS12) | (1 << CS10)); //CTC, prescaler = 1024
    TIMSK1 |= (1 << OCIE1A);                //Enable interrupt
    OCR1A = 16000;                    //CTC after 16000
                              // 4096000 Hz / 1024 = 4000 Hz = 1s --> 16000 = 4s

    sei(); //==>  SREG |= (1 << 7);     Enable interrupts!

    //INT0 DCF77 Init
    PORTD |= (1 << DDD2);                                 // Pull-Up wiederstand ON
    DDRD &= ~(1<<DDD2);                               //Port D2 as input

    //Relais (Monitor) Init
    DDRD |= (1<<DDD6);                           //D6 as output

    //INIT ADC
    ADMUX |= (0<<REFS1) | (1<<REFS0);       //Reference: AVcc with capacitor to GND at AREF.
    ADMUX |= (0<<MUX3) | (0<<MUX2) | (0<<MUX1) | (0<<MUX0); //ADC0
    ADCSRA |= (1<<ADPS2) | (1<<ADPS0);      //Frequenzvorteiler - 32
    ADCSRA |= (1<<ADEN);                  // ADC aktivieren
    DIDR0 = (1<<ADC0D);         //Digital input bei ADC0 deaktivieren.
    /*Dummy Readout*/
    readlight();            //Resultat auslesen
    /*Ende Dummy Readout*/

/*Initialitation ENDE ----------------------------------------------*/


/*Variablendefinition START */
    second = 0;
    time = 0;
    update = 1;
    int light;
    int oldlight=1;
/*Variablendefinition ENDE */
```

```
    dcffunction();
    PORTD |= (1 << DDD6);           //Relais ON
    _delay_ms(1000);
    uart_putsomething(hello);
    _delay_ms(5000);


    while(1) {              //Endlosschleife!

/*Ausgabe der Uhrzeit START----------------------------------------*/

        /* Display time (set = 0: no buttons legend, side = 0: show both hours and minutes) */
        if (update) puthour(time, format, 0, 0);

/*Ausgabe der Uhrzeit ENDE----------------------------------------*/

/*DCF77 Kontrolle*/
        if((time==3 || time==720) && second<=3)
        {
            PORTD &= ~(1 << DDD6);      //Relais OFF
            _delay_ms(100);
            dcffunction();
            PORTD |= (1 << DDD6);       //Relais ON
            _delay_ms(1000);
            }
/*Ende*/


/*Licht intensität auslesen und relais (monitor) ein/ausschalten*/
        light=readlight();
        if(light >= 930)
        {
            PORTD &= ~(1 << DDD6);          //Relais OFF
            oldlight=0;
            }

        else if(light < 930 && oldlight==0)
        {
            PORTD |= (1 << DDD6);           //Relais ON
            _delay_ms(1000);
            uart_putsomething(hello);
            _delay_ms(5000);
            oldlight=1;
            update=1;               //Uhrzeit wird angezeigt.
            }

/*Licht ENDE*/

        /* Button A + B pressed - open menu */
        _delay_ms(500);
        if (!(PINB & (1 << PINB0)) && !(PINB & (1 << PINB1))) menu();
    }

    return 0; //wird nie erreicht
}


ISR(TIMER1_COMPA_vect)      //Timer1 interrupt
{
    second++;
    if(second==15)
    {
        second = 0;
        time++;
        update=1;
        }
}
```

```c
/*puts.c*/
#include "puts.h"
/*
 _    _ _   ____            _   _ _   ______    _ _        _
| |  | (_) |  _ \          | | | (_) |  ____|  | | |      | |
| |__| |_  | |_) | __ _ ___| |_| |_  | |__ ___ | | | _____| |
|  __  | | |  _ < / _` / __| __| | | |  __/ _ \| | |/ / __| |
| |  | | | | |_) | (_| \__ \ |_| | | | | | (_) | |   <\__ \_|
|_|  |_|_| |____/ \__,_|___/\__|_|_| |_|  \___/|_|_|\_\___(_)*/

static unsigned char PROGMEM hello[]={
'\n','\n','\n','\n','\n','\n','\n','\n',' ','_',' ',' ',' ',' ','_',' ','_',' ',' ',' ','_','_','_','_',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','_',' ',' ',' ','_',' ','_',' ',' ',' ','_','_','_','_','_','_',' ',' ',' ',' ','_',' ','_',' ',' ',' ',' ',' ',' ',' ',' ','_','\n',
'|',' ','|',' ',' ','|',' ','(','_',')',' ','|',' ',' ','_',' ','\\',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','|',' ','|',' ','|',' ','(','_',')',' ','|',' ',' ','_','_','_','_','|',' ',' ','|',' ','|',' ','|',' ',' ',' ',' ',' ',' ','|',' ','|','\n',
'|',' ','|','_','_','|',' ','|','_',' ',' ','|',' ','|','_',')',' ','|',' ','_','_',' ','_',' ','_','_','_','|',' ','|','_','|',' ','|','_',' ',' ','|',' ','|','_','_',' ','_','_','_',' ','|',' ','|',' ','|',' ','_','_','_','_','_','|',' ','|','\n',
'|',' ',' ','_','_',' ',' ','|',' ','|',' ','|',' ',' ','_',' ','<',' ','/',' ','_','`',' ','/',' ','_','_','|',' ','_','_','|',' ','|',' ','|',' ','|',' ',' ','_','_','/',' ','_',' ','\\','|',' ','|',' ','|','/',' ','/',' ','_','_','|',' ','|','\n',
'|',' ','|',' ',' ','|',' ','|',' ','|',' ','|',' ','|','_',')',' ','|',' ','(','_','|',' ','\\','_','_',' ','\\',' ','|','_','|',' ','|',' ','|',' ','|',' ','|',' ','|',' ','(','_',')',' ','|',' ','|',' ',' ',' ','<','\\','_','_',' ','\\','_','|','\n',
'|','_','|',' ',' ','|','_','|','_','|',' ','|','_','_','_','_','/',' ','\\','_','_',',','_','|','_','_','_','/','\\','_','_','|','_','|','_','|',' ','|','_','|',' ',' ','\\','_','_','_','/','|','_','|','_','|','\\','_','\\','_','_','_','(','_',')','\n','\n','\n','\n','\n','\n','\n','\n','\n','\n','\0'};

static unsigned char PROGMEM AM[] =
"\n                                   0000  00 00\n                                   0  0  0 0 0
\n                                   0000  0   0\n                                   0  0  0   0
\n                                   0  0  0   0\n";

static unsigned char PROGMEM PM[] =
"\n                                   0000  00 00\n                                   0  0  0 0 0
\n                                   0000  0   0\n                                   0     0   0
\n                                   0     0   0\n";


static unsigned char PROGMEM digits[10][5 * 16] = {
    { // 0
        4, 11,  4,  0,  0,
        2, 15,  2,  0,  0,
        1, 17,  1,  0,  0,
        1, 17,  1,  0,  0,
        1,  7,  3,  7,  1,
        1,  6,  5,  6,  1,
        1,  6,  5,  6,  1,
        1,  6,  5,  6,  1,
        1,  6,  5,  6,  1,
        1,  6,  5,  6,  1,
        1,  6,  5,  6,  1,
        1,  7,  3,  7,  1,
        1, 17,  1,  0,  0,
        1, 17,  1,  0,  0,
        2, 15,  2,  0,  0,
        4, 11,  4,  0,  0
    },
    { // 1
        8,  7,  4,  0,  0,
        7,  8,  4,  0,  0,
        6,  9,  4,  0,  0,
        5, 10,  4,  0,  0,
        4, 11,  4,  0,  0,
        4, 11,  4,  0,  0,
        9,  6,  4,  0,  0,
        9,  6,  4,  0,  0,
        9,  6,  4,  0,  0,
        9,  6,  4,  0,  0,
        9,  6,  4,  0,  0,
```

```
            9,  6,  4,  0,  0,
            9,  6,  4,  0,  0,
            9,  6,  4,  0,  0,
            9,  6,  4,  0,  0,
            9,  6,  4,  0,  0
        },
        { // 2
            4, 11,  4,  0,  0,
            2, 15,  2,  0,  0,
            1, 17,  1,  0,  0,
            1, 17,  1,  0,  0,
            1,  7,  4,  6,  1,
           12,  6,  1,  0,  0,
           10,  8,  1,  0,  0,
            8, 10,  1,  0,  0,
            6, 10,  3,  0,  0,
            4, 10,  5,  0,  0,
            2, 10,  7,  0,  0,
            1,  9,  9,  0,  0,
            1, 17,  1,  0,  0,
            1, 17,  1,  0,  0,
            1, 17,  1,  0,  0,
            1, 17,  1,  0,  0
        },
        { // 3
            4,11,4,0,0,
            2,15,2,0,0,
            1,17,1,0,0,
            1,17,1,0,0,
            1,7,4,6,1,
            12,6,1,0,0,
            8,10,1,0,0,
            8,8,3,0,0,
            8,8,3,0,0,
            8,10,1,0,0,
            12,6,1,0,0,
            1,7,4,6,1,
            1,17,1,0,0,
            1,17,1,0,0,
            2,15,2,0,0,
            4,11,4,0,0
        },
        { // 4
            9,7,3,0,0,
            8,8,3,0,0,
            7,9,3,0,0,
            6,10,3,0,0,
            5,11,3,0,0,
            4,12,3,0,0,
            3,6,1,6,3,
            2,6,2,6,3,
            1,6,3,6,3,
            1,17,1,0,0,
            1,17,1,0,0,
            1,17,1,0,0,
            1,17,1,0,0,
            10,6,3,0,0,
            10,6,3,0,0,
            10,6,3,0,0,
        },
        { // 5
            1,17,1,0,0,
            1,17,1,0,0,
            1,17,1,0,0,
            1,17,1,0,0,
            1,6,12,0,0,
            1,6,12,0,0,
            1,14,4,0,0,
            1,16,2,0,0,
            1,17,1,0,0,
            12,6,1,0,0,
            12,6,1,0,0,
            1,6,5,6,1,
            1,17,1,0,0,
```

```
            1,17,1,0,0,
            2,15,2,0,0,
            4,11,4,0,0,
        },
        { // 6
            8,7,4,0,0,
            7,7,5,0,0,
            6,7,6,0,0,
            5,7,7,0,0,
            4,7,8,0,0,
            3,7,9,0,0,
            2,7,10,0,0,
            1,14,4,0,0,
            1,16,2,0,0,
            1,17,1,0,0,
            1,6,5,6,1,
            1,6,5,6,1,
            1,17,1,0,0,
            1,17,1,0,0,
            2,15,2,0,0,
            4,11,4,0,0,
        },
        { // 7
            1,17,1,0,0,
            1,17,1,0,0,
            1,17,1,0,0,
            1,17,1,0,0,
            11,7,1,0,0,
            10,7,2,0,0,
            9,7,3,0,0,
            8,7,4,0,0,
            7,7,5,0,0,
            6,7,6,0,0,
            5,7,7,0,0,
            4,7,8,0,0,
            3,7,9,0,0,
            2,7,10,0,0,
            1,7,11,0,0,
            1,6,12,0,0
        },
        { // 8
            4,11,4,0,0,
            2,15,2,0,0,
            1,17,1,0,0,
            1,17,1,0,0,
            1,6,5,6,1,
            1,6,5,6,1,
            1,17,1,0,0,
            3,13,3,0,0,
            3,13,3,0,0,
            1,17,1,0,0,
            1,6,5,6,1,
            1,6,5,6,1,
            1,17,1,0,0,
            1,17,1,0,0,
            2,15,2,0,0,
            4,11,4,0,0
        },
        { // 9
            4,11,4,0,0,
            2,15,2,0,0,
            1,17,1,0,0,
            1,17,1,0,0,
            1,6,5,6,1,
            1,6,5,6,1,
            1,17,1,0,0,
            1,17,1,0,0,
            3,14,2,0,0,
            9,7,3,0,0,
            8,7,4,0,0,
            7,7,5,0,0,
            6,7,6,0,0,
            5,7,7,0,0,
            4,7,8,0,0,
```

```
        3,7,9,0,0,
 }
};

void uart_putsomething(char *something) {
    int symbols=0;
    int i;
    for(i=0; pgm_read_byte(&something[i])!='\0'; i++)
    {
        if(pgm_read_byte(&something[i])=='\n')          //Der Monitor kennt \n (neue Zeile) nicht. Also wird bis zum 80 Symbol mit ' ' gefüllt
        {
            int j;
            for(j=0;j<80-symbols;j++)
            {
                uart_putc(' ');
                }
            symbols=0;
            }
        else
        {
            uart_putc(pgm_read_byte(&something[i]));
            symbols++;
            }
        }
}

int uart_putc(unsigned char c) {
    while (!(UCSR0A & (1<<UDRE0))) {}
    UDR0 = c;
    return 0;
}

void uart_puts(char *s) {
    while (*s) {            // so lange *s != '\0' also ungleich dem "String-Endezeichen"
        uart_putc(*s);
        s++;
    }
}

/**
 * Display hour from single digits value (called from puthour, which takes timeutes from 0 to 1440).
 * FOR COMPRESSED DIGITS
 *
 * @param h_l The left hours digit
 * @param h_r The right hours digit
 * @param m_l The left timeutes digit
 * @param m_r The right timeutes digit
 * @param side Tell which side display (0: both, 1: left, 2: right)
 */
void uart_puthour(int h_l, int h_r, int m_l, int m_r, int side) {

    /* Row-column indices */
    int i, j, k;

    /* Column loop: put i-line of each 4 digit side-by-side */
    for (i = 0; i < 16; i++) {

            /* Row loop 1: put j char of i-line of left hour */
            for (j = 0; j < 5; j++) {

                /* Digit loop: decompress charachters */
                for (k = 0; k < pgm_read_byte(&digits[h_l][j + 5 * i]); k++) {
                    if (side == 2) uart_putc(' '); /* Display only if not modifing timeute in set_hour */
                    else {
                        if (j % 2) uart_putc('0'); /* If index odd put '0' */
                        else uart_putc(' '); /* If index even put '1' */
                    }
                }
            }

        /* Row loop 2: put j char of i-line of right hour */
```

```
276         for (j = 0; j < 5; j++) {
277 
278             /* Digit loop: decompress charachters */
279             for (k = 0; k < pgm_read_byte(&digits[h_r][j + 5 * i]); k++) {
280                 if (side == 2) uart_putc(' '); /* Display only if not modifing timeute in
    set_hour */
281                 else {
282                     if (j % 2) uart_putc('0'); /* If index odd put '0' */
283                     else uart_putc(' '); /* If index even put '1' */
284                 }
285             }
286         }
287 
288         /* Draw two dots between hours and timeutes */
289         if (i == 5 || i == 6 || i == 10 || i == 11) uart_puts(" 00 ");
290         else uart_puts("    ");
291 
292         /* Row loop 3: put j char of i-line of left timeute */
293         for (j = 0; j < 5; j++) {
294 
295             /* Digit loop: decompress charachters */
296             for (k = 0; k < pgm_read_byte(&digits[m_l][j + 5 * i]); k++) {
297                 if (side == 1) uart_putc(' '); /* Display only if not modifing timeute in
    set_hour */
298                 else {
299                     if (j % 2) uart_putc('0'); /* If index odd put '0' */
300                     else uart_putc(' '); /* If index even put '1' */
301                 }
302             }
303         }
304 
305         /* Row loop 4: put j char of i-line of right timeute */
306         for (j = 0; j < 5; j++) {
307 
308             /* Digit loop: decompress charachters */
309             for (k = 0; k < pgm_read_byte(&digits[m_r][j + 5 * i]); k++) {
310                 if (side == 1) uart_putc(' '); /* Display only if not modifing timeute in
    set_hour */
311                 else {
312                     if (j % 2) uart_putc('0'); /* If index odd put '0' */
313                     else uart_putc(' '); /* If index even put '1' */
314                 }
315             }
316         }
317     }
318 }
319 
320 /**
321  * Display hour (calls uart_puthour).
322  *
323  * @param format The format of displayed time (0: 24h, 1: 12h)
324  * @param set Tell to the function if display in set mode (0: normal, 1: adds buttons legend)
325  * @param side Tell to uart_puthour which side display (if set is 1)
326  */
327 void puthour(int format, int set, int side) {
328 
329     /* Set time to 0 at midnight */
330     if (time >= 1440) time = 0;
331 
332     /* If set mode show buttons legend */
333     if (!set) uart_puts("\n\n");
334     else uart_putsomething(menu_set_hour);
335     if (!format) uart_puts("\n\n\n");
336 
337     /* Call uart_puthour dividing time into 4 digits */
338     uart_puthour( time / 600 - 1 * (format && ((time / 60) > 12)) - 1 * (format && (time / 60)
    >= 20 && (time / 60) <= 21), /* Hour left */
339                   ((time / 60) + 8 * (format && ((time / 60) > 12))) % 10, /* Hour right */
340                   (time / 10) % 6, /* time left */
341                   time % 10, /* time right */
342                   side ); /* Side to show (for set_hour()) */
343 
344     /* If 12h mode show am or pm */
345     if (format) {
```

```
346             if ((time / 60) < 12) uart_putsomething(AM);        //AM
347             else uart_putsomething(PM);         //PM
348         }
349         else uart_puts("\n\n\n");
350 
351         /* */
352         update = 0;
353     }
```

```
/*dcf.c*/
#include "dcf.h"
/*Signal Beispiel (10ms polling (3+...)x100 pol = 3+... s):
1111111111111111111111110000000000111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
                          |STEP ONE im Fall, dass hier (irgendwo) angefangen wird, wird 75ms
gewartet | bis hier etwa.
0000000001111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111000000000
|Signal Anfang (1 mal pro sekunde), falls dauer 10 ms =~ 10 abtastungen Logisch 0. dauer 20ms Logisch
1.
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
              Kein signal in dieser Sekunde = letzte Sekunde der Minute.|
1111111111111111111111111111111111111111111111110000000000000000000001111111111111111111111111111111111111111111111111111111111111
                                                    |Beginn neuer Minute immer mit 20ms Signal = logisch
1.
*/
int dcffunction()
{
    int pointer=0;
    int ones=0;
    int zeros=0;
    int dcfarray[59];
    int bit=1;
    int temp=0;
    int errors=0;

/*STEP ONE: Im Fall, dass mitten im Signal angefangen wird zu pollen, wird gepollt, bis das erste "0"
erscheint und dann 70ms gewartet. Damit stellt man sicher, dass beim STEP TWO eine ganze Sekunde
abgefangen wird.*/
    while(bit==1)                       //Wait until first signal (bit=0)
    {
        if(errors>=1000) return 0;      //Kein Signalempfang!
        _delay_ms(10);      //wait 10ms
        if ( PIND & (1<<2) ) bit=1; else bit=0; //Read input status 1=no signal | 0=signal.
        errors++;
        }
    _delay_ms(75);          //wait 75ms
    errors=0;           //Falls verspätetes Signal Fehler auf 0 setzen.

/*STEP TWO: Es wird gepollt, bis zum Anfang des Signals. Danach wird ausgewertet, ob speichern,
aufhören, weitermachen. Eine "Sekunde" dauert vom Anfang eines Signals, bis zum Anfang des nächsten.*/
    while(1)
    {
        _delay_ms(10);
        if(ones>=1000) return 1;        //Empfang verloren.
        if ( PIND & (1<<2) ) ones++; //Falls kein signal "1" werden einfach die "1" gezählt.
        else                                            //Sobald signal "0" kommt (Anfang nächstes
Signal/neue Sekunde)
        {
            if(ones > 30 && ones <= 130)        //Falls mehr als 30 "1" (Also nicht nach STEP ONE),
und weniger als 100 (Noch keine Minute Vorbei) wird ein Bit gespeichert, je nach Anzahl
"0".
            {
                if(zeros>=5 && zeros <=15) dcfarray[pointer++]=0;       //Logisches 0
                else if(zeros >15 && zeros <=30) dcfarray[pointer++]=1;     //Logisches 1
                zeros=ones=0;
                }
            else if(ones > 130)         //Falls mehr als 100 "1", Minute vorbei.
            {
                if(pointer==58)                             //Minute komplett aufgezeichnet
                {
                    temp=second;                            //Sekundenanzahl von Timer speichern und
auf 0 setzen
                    second=0;
                    if(!evalbits(dcfarray)) goto error;     //STEP THREE falls parity bits ok, weiter,
sonst FEHLER.
                    TCNT1=0;                                //Alles auf 0 setzen
                    second=0;
                    update=1;                               //update display? YES!
                    return 1;                               //OK!
                    }
                else                                        //Falls Minute nicht komplett oder FEHLER
                {
                    error:
                    if(errors>=3) {update=1; return 0;}         //Keine gute Kommunikation mit DCF77
```

```
möglich.
                    second=temp;                            //Timer-Sekunden wieder zurücksetzen
(damit falls zu oft Fehler auftreten die Uhr trotzdem weiterläuft.
                    pointer=0;
                    ones=zeros=0;
                    errors++;
                    }
                }
            zeros++;                    //Sonst wird einfach die Anzahl "0" inkrementiert.
            }
        }

}

int evalbits(int array[])
{
    int minutes = 0;
    int hours = 0;
    int parity = 0;
    if(array[21]==1) {minutes=minutes+1; ++parity;}             //Evalue minutes bits
    if(array[22]==1) {minutes=minutes+2; ++parity;}
    if(array[23]==1){minutes=minutes+4; ++parity;}
    if(array[24]==1) {minutes=minutes+8; ++parity;}
    if(array[25]==1) {minutes=minutes+10; ++parity;}
    if(array[26]==1) {minutes=minutes+20; ++parity;}
    if(array[27]==1) {minutes=minutes+40; ++parity;}
    if(((parity % 2 != 0) && (array[28]==0)) || ((parity % 2 == 0) && (array[28]==1))) return 0;
    parity=0;
    if(array[29]==1) {hours=hours+1; ++parity;}             //Evalue hours bits
    if(array[30]==1) {hours=hours+2; ++parity;}
    if(array[31]==1) {hours=hours+4; ++parity;}
    if(array[32]==1) {hours=hours+8; ++parity;}
    if(array[33]==1) {hours=hours+10; ++parity;}
    if(array[34]==1) {hours=hours+20; ++parity;}
    if(((parity % 2 != 0) && (array[28]==0)) || ((parity % 2 == 0) && (array[28]==1))) return 0;    //
If number of "1" bits is even but the parity bit is 1 or if number of "1" bits is odd but the parity
bit is 0 TRANSMISSION ERROR.

    time=hours*60+minutes;          //Write new hour on the variable
    return 1;
    }
```

```c
/*light.c*/
#include "light.h"

int readlight()
{
    ADCSRA |= (1<<ADSC);            // eine Wandlung "single conversion"
    while (ADCSRA & (1<<ADSC) ) {}  // auf Abschluss der Konvertierung warten
    return ADC;          //Resultat auslesen
}
```

```c
/* menu.c */
#include "menu.h"

/* Menu texts */
static unsigned char PROGMEM menu_title[881] =
"
#######     ###    #       #    ###    #########    #####   #      #            #      #   #
#   #       #   #   #       #       #     #  #      #            #      #  #     #  #       #
#     #      #      #         #      #            #######   #######  #   #   #  #######
#      #         ########            #      #  #     #  #   #   #  #     #      #      #
#      #            #      #  #     #   #  #  #   #     #      #       #     #  #
#            #######   #     #    ## ##    #     #      #        #####   #
#
v2.9.1      \0";
static unsigned char PROGMEM menu_main[1121] =
"
Menu:
- Button A: regulate
time
- Button B: choose time
format
- Buttons A + B:
return
Soft- & Hardware: Jonas Pfaff & Fabio
Banfi
Original project: Mario
Mauerer
\0";
static unsigned char PROGMEM menu_set_hour[241] =
"
A: Increase  --  B:
Next
\0";
static unsigned char PROGMEM menu_set_format[1121] =
"
Format:
- Button A: 24h
mode
- Button B: 12h
mode
- Buttons A + B:
return
Soft- & Hardware: Jonas Pfaff & Fabio
Banfi
Original project: Mario
Mauerer
\0";

/**
  * Increase single digit by one pressing button A starting from the left one.
  * Pressing button B will be modified next digit right.
  * When last digit right is reached, pressing B function will return.
  * Note: display time only in 24h format.
  */
void set_hour(void) {

    /* Put text */
    puthour(time, 0, 1, 1); /* Set hour only in 24h format allowed, show only left side */

    /* 1: modify hours, 2: modify minutes */
    int pos = 1;

    /* Endless loop */
    while (1) {

        /* Wait for a while to get new buttons state */
        /****************** WAIT ******************/

        /* Button A pressed */
        if (!(PINB & (1 << PINB0))) {

            switch (pos) {

                case 1: /* Hour */
```

```
                    time += 60; /* Simply add an hour */
                    break;

                case 2: /* Minute */
                    if (time % 60 != 59) time++; /* Pressing A at 59 minutes must not increase time by 1 hour */
                    else time -= 59; /* Instead decrease by 59 */
                    break;
            }

            /* Round time */
            time = time % 1440;

            /* Show new hour */
            puthour(time, 0, 1, pos);
        }

        /* Button B pressed */
        if (!(PINB & (1 << PINB1))) {

            /* If B pressed first time modify minutes, return when button B pressed second time */
            if (++pos == 3) {
                puthour(time, format, 0, 0);
                break;
            }

            /* Show new hour */
            puthour(time, 0, 1, pos);
        }
    }
}

/**
  * Select the displayed hour format:
  * - pressing button A format is set to 24h format, then function will return.
  * - pressing button B format is set to 12h format, then function will return.
  * If button A and B are pressed together function will return and format will be as defined before call.
  */
void set_format(void) {

    /* Put text */
    uart_putsomething(menu_title);
    uart_putsomething(menu_set_format);

    /* Wait for a while to get new buttons state */
    /******************** WAIT ********************/

    /* Endless loop */
    while (1) {

        /* Button A pressed */
        if (!(PINB & (1 << PINB0))) {
            format = 0;
            break;
        }

        /* Button B pressed */
        if (!(PINB & (1 << PINB1))) {
            format = 1;
            break;
        }

        /* Button A + B pressed */
        /******************** WAIT ********************/
        if (!(PINB & (1 << PINB0)) && !(PINB & (1 << PINB1)))
        {
            puthour(time, format, 0, 0);
            break;
            }
    }
}

```

```
110  /**
111    * Modify BAWATCH parameters:
112    * - pressing button A will be possible to set hour.
113    * - pressing button B will be possible to choose between 24h or 12h display format.
114    * If button A and B are pressed together function will return and format will be as defined
     before call.
115    * Function return anyway after hour or format are set.
116    */
117  void menu(void) {
118
119      /* Put text */
120      uart_putsomething(menu_title);
121      uart_putsomething(menu_main);
122
123      _delay_ms(500);
124
125      /* Endless loop */
126      while (1) {
127
128          /* Button A pressed */
129          if (!(PINB & (1 << PINB0))) {
130              set_hour();
131              return;
132          }
133
134          /* Button B pressed */
135          if (!(PINB & (1 << PINB1))) {
136              set_format();
137              return;
138          }
139
140          /* Button A + B pressed */
141          _delay_ms(500);
142          if (!(PINB & (1 << PINB0)) && !(PINB & (1 << PINB1))) break;
143      }
144
145      /* Show hour */
146      puthour(time, format, 0, 0);
147  }
```