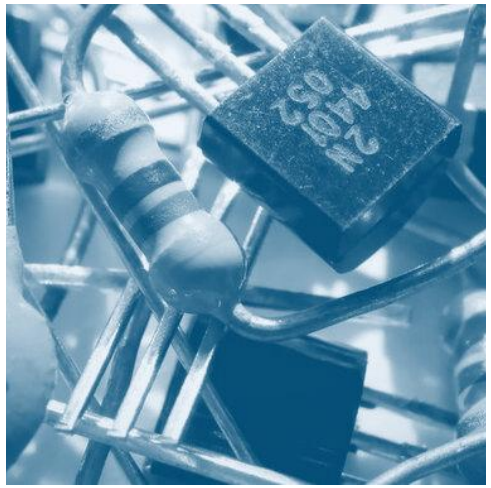
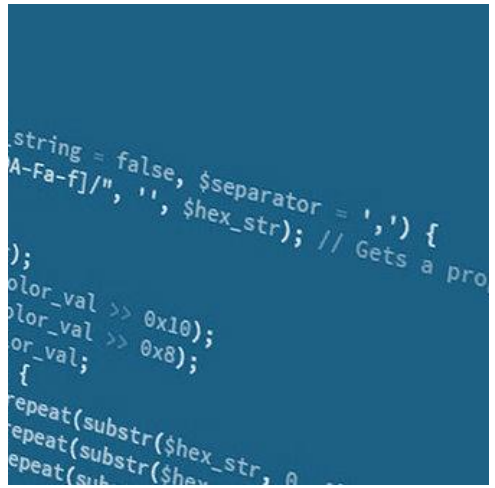
BINÄRE UHR

Funktionen für den Benutzer



Bauelemente und Konfiguration



Programmierung und Messung

INHALT

FUNKTIONEN FÜR DIE BENUTZER

Tasten

- Taste1: Steuerung der Helligkeit 🔆
- Taste2: Aktivierung/ Deaktivierung des Sleep Modus 💤
- Taste3: Off/On und Speicherung der Zustand.

Taste 1

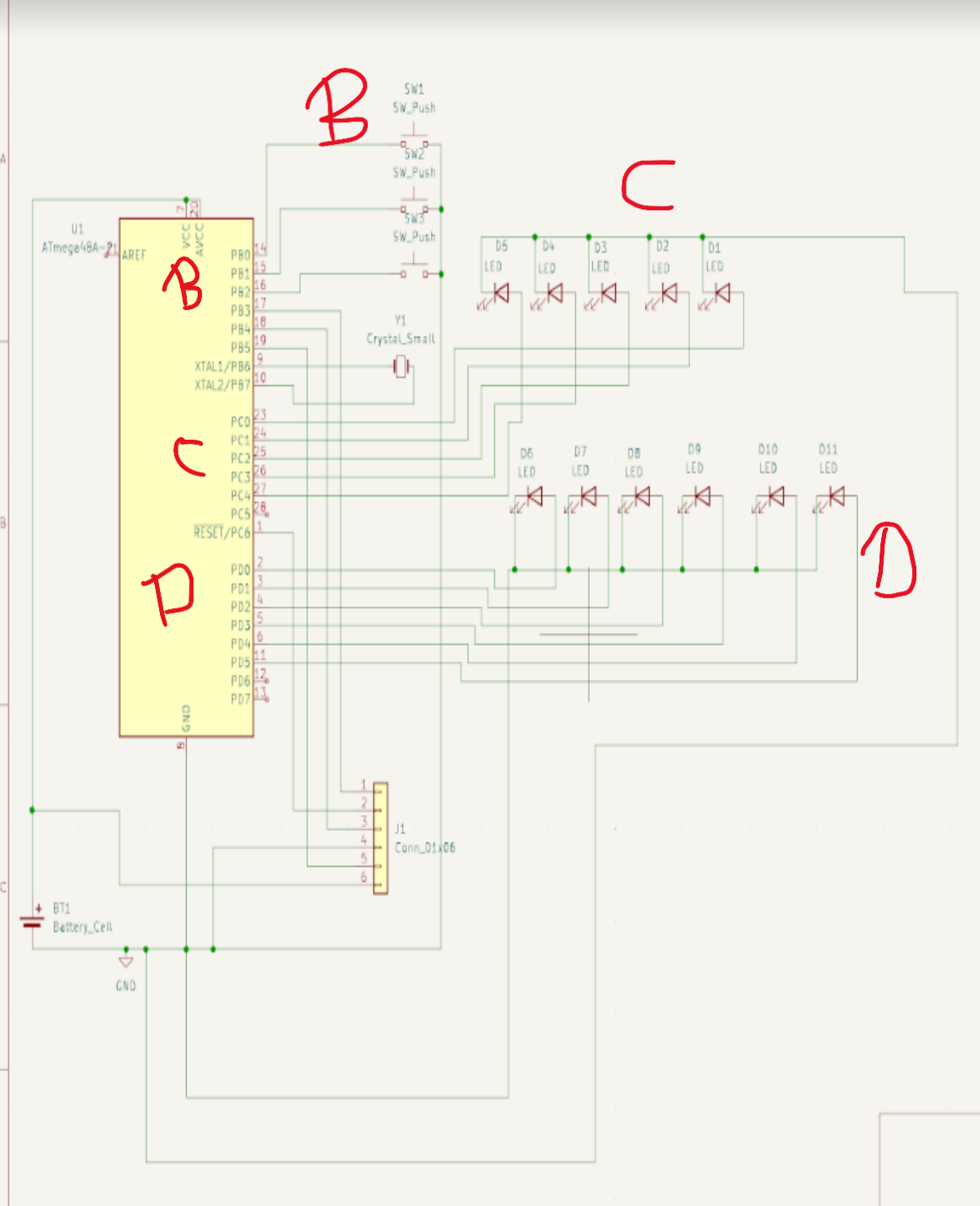
Taste 2

- Taste 1&2 : Sekunden- /Minuten Modus.
- Taste 1&3 : Erhöhung der Stunden
- Taste 2&3 : Ehöhung der Minuten

Taste 3

BAUELEMENTE

- Atmega 48A
- LEDs: Port D Minute und Port C Stunden.
- Tasten: Port B PCINT
- Uhrenquarz (32 768 Hz) ASC PIN9/10



PROGRAMMIERUNG

- Globale variable:

```
volatile uint16_t hours ;
volatile uint16_t minutes ;
volatile uint16_t seconds ;
volatile int8_t brightness = 3;
volatile uint8_t powerSaveMode = 0;
volatile uint8_t powerDownMode = 0;
volatile uint8_t sec_Mode = 0;
```

PROGRAMMIERUNG

- Wichtige Funktionen:

```
void initLEDs();

void initTaste();

void init_PWM();
void enable_LEDs();
void disable_LEDs();

void initTimer();

void saveToEEPROM();
void loadFromEEPROM();
```

MAIN/PROGRAMMSTART

Die LEDs als Output

```
void initLEDs(){
    // Initialisiere Portrichtungen
    DDRD |= 0x3F; // Setze PD0-PD5 als Ausgänge für Minuten
    DDRC |= 0x1F; // Setze PC0-PC4 als Ausgänge für Stunden
    PORTD &= 0x3F; // Setze die Ausgänge für Minuten auf 0
    PORTC &= 0x1F; // Setze die Ausgänge für Stunden auf 0
//  DDRD |= (1 << PD7); // Setze PD7 als Ausgang für die LED
//  PORTD &= (1 << PD7); // Setze die LED auf aus
}
```

```
int main(void) {
    // Aktualisiere die Werte im EEPROM
    loadFromEEPROM();

    initLEDs();
    initTaste();

    init_PWM();

    initTimer();

    sei(); // Enable global interrupts

    while (1) {
        if (powerDownMode) {
            set_sleep_mode(SLEEP_MODE_PWR_DOWN);
            sleep_enable();
            sleep_cpu(); // Enter Power-Down Mode
        }
        else if (powerSaveMode) {
            set_sleep_mode(SLEEP_MODE_PWR_SAVE);
            sleep_enable();
            sleep_cpu(); // Enter Power-Save Mode
        }
        else {
            sleep_disable();
            init_PWM();
        }
    }


    return 0;
```

MAIN/PROGRAMMSTART

```
int main(void) {
    // Aktualisiere die Werte im EEPROM
    loadFromEEPROM();

    initLEDs();
    initTaste();

    initTimer();

    init_PWM();



    sei(); // Enable global interrupts
```

Die Tasten als digital Input

```
void initTaste(){
    // Initialisiere Taster
    DDRB &= ~((1 << PB0) | (1 << PB1) | (1 << PB2)); // Set PB0, PB1, and PB2 as inputs for buttons
    PORTB |= (1 << PB0) | (1 << PB1) | (1 << PB2); // Enable internal pull-up resistors for buttons
    PCMSK0 |= (1 << PCINT0) | (1 << PCINT1) | (1 << PCINT2); // Aktiviere PCINT0, PCINT1 und PCINT2 für Pin Change Interrupt für PB0, PB1, PB2
    PCICR |= (1 << PCIE0); // Aktiviere PCIE0 für PCINT0, PCINT1 und PCINT2


}
```

TIMER2
FÜR DIE SEKUNDEN BZW. MIN. STUNDEN.

```
void initTimer() {
    ASSR |= (1 << AS2); // Asynchroner Modus für Timer 2 aktivieren
    TCCR2A |= (1 << WGM21); // CTC-Modus (Clear Timer on Compare Match) aktivieren
    TCCR2B |= (1 << CS22) | (1 << CS20); // Prescaler auf 128 setzen, Timer starten
    OCR2A = 255; // Vergleichswert für ~1-Sekunden-Interrupt setzen
    // Compare Match Interrupt für Timer 2 aktivieren Interrupt ausgelöst, wenn der Timer OCR2A erreicht.
    TIMSK2 |= (1 << OCIE2A);
}
```

$$f_{OCnx} = \frac{f_{clk\_I/O}}{2 \times N \times (1 + OCRnx)}$$

$$f_{OCnx} \times 2 \times N \times (1 + OCRnx) = f_{clk\_I/O}$$

$$f_{OCnx} \times 2 \times N + f_{OCnx} \times 2 \times N \times OCRnx = f_{clk\_I/O}$$

$$f_{OCnx} \times 2 \times N \times OCRnx = f_{clk\_I/O} - f_{OCnx} \times 2 \times N$$

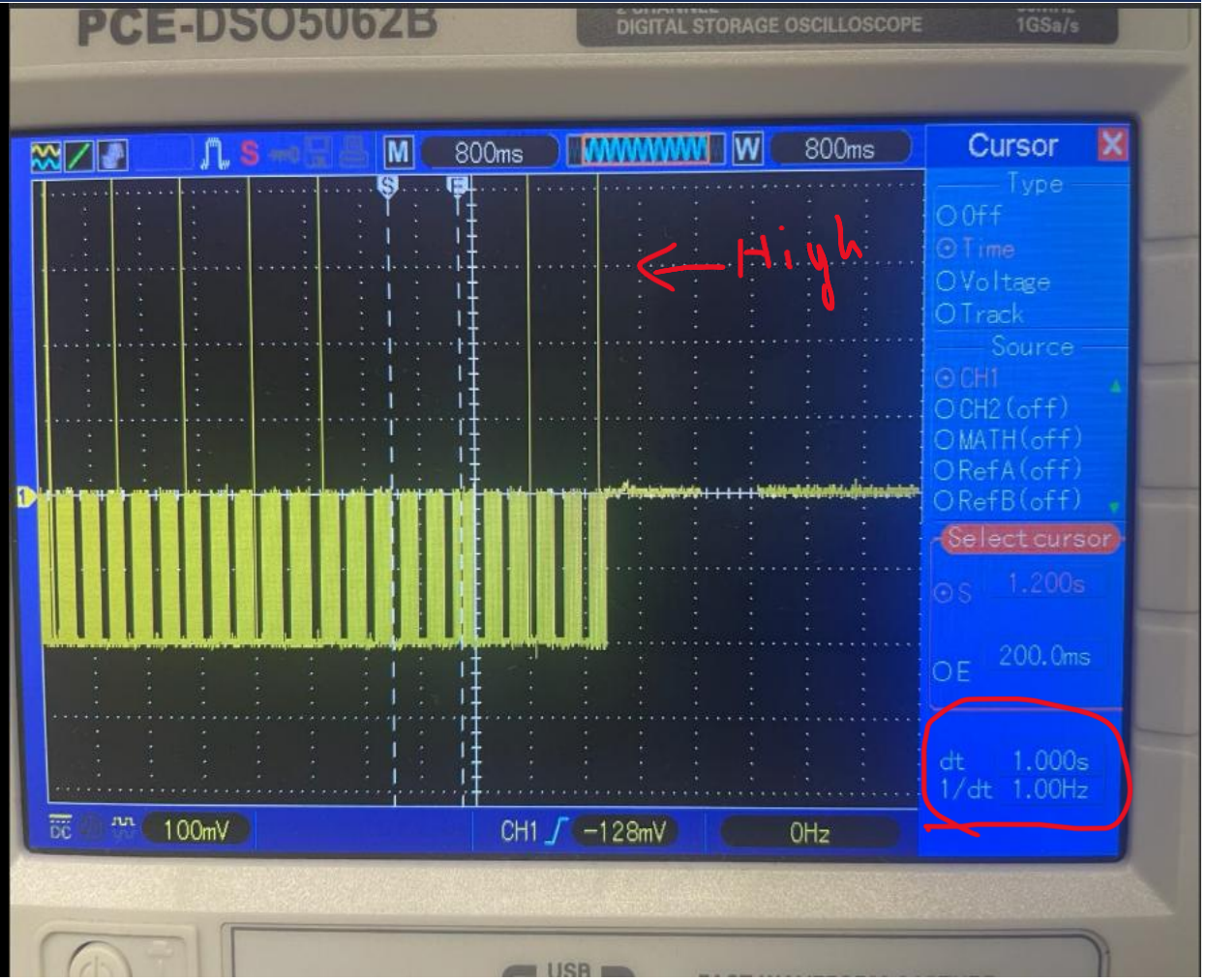
$$OCRnx = \frac{f_{clk\_I/O} - f_{OCnx} \times 2 \times N}{f_{OCnx} \times 2 \times N}$$

```
ISR(TIMER2_COMPA_vect) {
    seconds++;
    if (seconds == 60) {
        seconds = 0;
        minutes++;
        if (minutes == 60) {
            minutes = 0;
            hours++;
            if (hours == 24) {
                hours = 0;
            }
        }
    }
//  PORTD ^= (1 << PD7); // Toggle LED
}
```

TIMER2
RICHTIGKEIT MESSUNG

AVR Timer Calculator

System Clock Frequency (Hz):	32768	
Timer Resolution:	16 bit	Calculate Using ...
Prescaler:	(5) Clk/128	
Total Timer Ticks:	256	... Total Timer Ticks
Overflow Count:	0	... Overflows and Remainder
Remainder Timer Ticks:	256	
Real Time (sec):	1	... Real Time
New Freq (Hz):	1	... New Freq



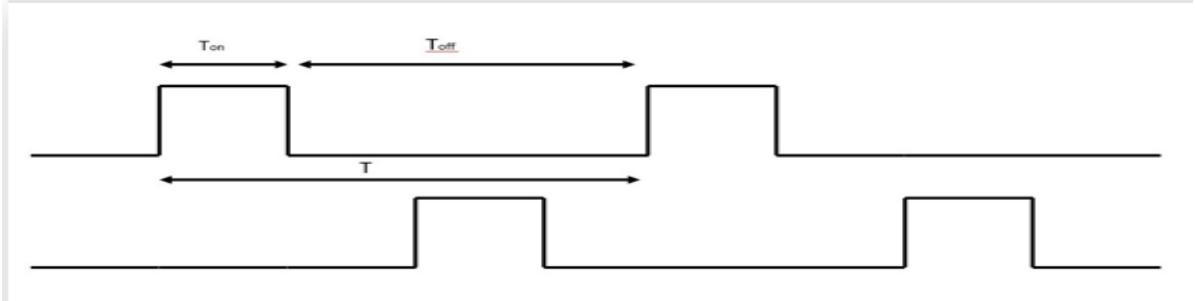
HELLIGKEIT

```
void init_PWM() {
    //Aktiviert die Timer0 Compare Interrupts für OC0A und OC0B.
    TIMSK0 |= (1 << OCIE0B) | (1 << OCIE0A);
    //Wählt den Timer0 Clock mit Prescaler 1 (CS00) und den ctc (WGM01).
    TCCR0B  |= (1 << CS00) | (1 << WGM01);

    // Begrenze die Helligkeit auf den Bereich von -3 bis 3
    if (brightness > 3) {
        brightness = 3;
        } else if (brightness < -3) {
        brightness = 3;//wiederhole die Runde
    }

    // Berechnung der PWM-Duty-Cycle-Werte basierend auf Helligkeitswert
    OCR0A = 126 + (50 * brightness);
    OCR0B = 126 - (50 * brightness); // OCR0B ist das Komplement von OCR0A
}
```

```
ISR(TIMER0_COMPA_vect) {
    enable_LEDs();
}

ISR(TIMER0_COMPB_vect) {
    disable_LEDs();
}
```

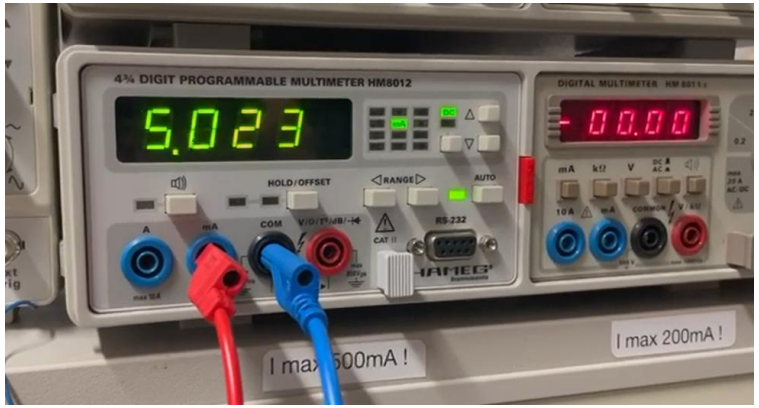
```
void disable_LEDs() {
    PORTD = 0;
    PORTC = 0;
}
```

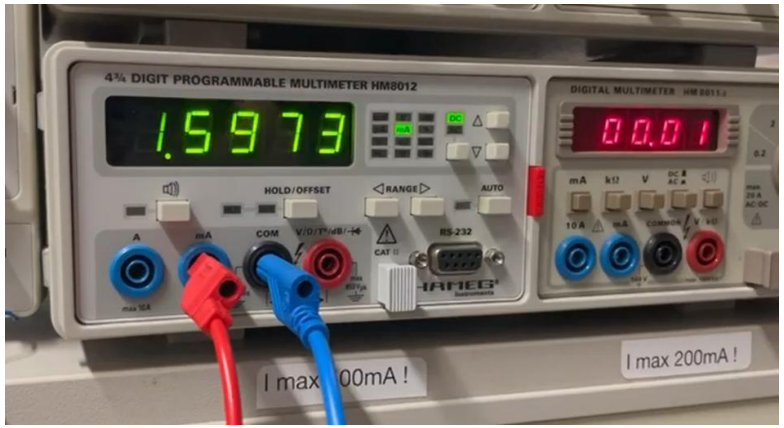
```
oid enable_LEDs() {
    PORTC = hours;

    if(sec_Mode){
        PORTD = seconds;
    }
    else{
        PORTD= minutes;
    }
}
```

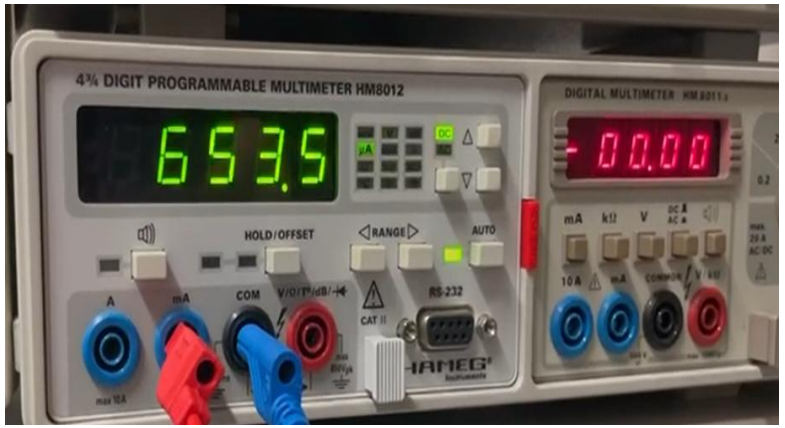


Duty Cycle = Ton / Ton + Toff

```
//  Aktionen für Tastenkombinationen in Gruppe 1 aus
if (!(PINB & (1 << PB0)) && !powerSaveMode && !powerDownMode) {
    _delay_ms(20); // Stabilisierung
    while (!(PINB & (1 << PB0))); // Warte auf release
    brightness = brightness - 3;
    init_PWM();
}
```







VERBRAUCH BEI ÄNDERUNG DER HELLIGKEIT ☼

INTERRUPTS

- Nach der Initialisierung werden die Interrupts aktiviert
- PCINT0-2 Für BP0-1
- Timer2/0

```
sei(); // aktiviere globale interrupts
```

BUTTONS
IN 2 GRUPPEN AUFTEILEN

Gruppe 1 einzusetze Taste

```
ISR(PCINT0_vect) {
        _delay_ms(10);// um mit der Geschwindigkeit der User umzugehen

        //Falls eine einzehlne Taste gedruckt wird
        if ((PINB & 0b00000111) == 0b00000110 ||
        (PINB & 0b00000111) == 0b00000101 ||
        (PINB & 0b00000111) == 0b00000011){
        // Gruppe 1: Tastenkombinationen, die in Gruppe 1 gehören
        //  Aktionen für Tastenkombinationen in Gruppe 1.
            if (!(PINB & (1 << PB0)) && !powerSaveMode && !powerDownMode) {
                _delay_ms(20); // Stabilisierung
                while (!(PINB & (1 << PB0))); // Warte auf release
                brightness = brightness - 3;
                init_PWM();
            }

            if (!(PINB & (1 << PB1)) && !powerDownMode) {
                _delay_ms(20); // Stabilisierung
                while (!(PINB & (1 << PB1))); // Wait for button release
                powerSaveMode = !powerSaveMode;
            }

            if (!(PINB & (1 << PB2))) {
                _delay_ms(20); // Stabilisierung
                while (!(PINB & (1 << PB2))); // Wait for button release
                powerDownMode = !powerDownMode;
                if (powerDownMode) {
                    saveToEEPROM();
                }
            }
    }
```

Gruppe 2 Tasten gleichzeitig

```
    //Falls 2 Tasten zusammen gleichzeitig Gedruckt werden
    else {
        // Gruppe 2: Tastenkombinationen, die in Gruppe 2 gehören
        // Aktionen für Tastenkombinationen in Gruppe 2.
        if (!(PINB & ((1 << PB0) | (1 << PB1)))) {
            _delay_ms(20); // Stabilisierung
            while (!(PINB & ((1 << PB0) | (1 << PB1)))); // Wait for button release
            sec_Mode = !sec_Mode;
            init_PWM();
        }

        if (!(PINB & ((1 << PB1) | (1 << PB2)))) {
            _delay_ms(20); // Stabilisierung
            while (!(PINB & ((1 << PB1) | (1 << PB2)))); // Wait for button release
            ++minutes;
            if(minutes==60){
                minutes=0;
            }
            //init_PWM();
        }

        if (!(PINB & ((1 << PB0) | (1 << PB2)))) {
            _delay_ms(20); // Stabilisierung
            while (!(PINB & ((1 << PB0) | (1 << PB2)))); // Wait for button release
            ++hours;
            if(hours==24){
                hours=0;
            }
            //init_PWM();
        }
    }
}
```

BP0/PCINT0 /TASTE1
STEUERUNG DER HELLIGKEIT

Globale Änderung

```
volatile int8_t brightness = 3;
```

in ISR(PCINT0_vect)

```
if (!(PINB & (1 << PB0)) && !powerSaveMode && !powerDownMode) {
    _delay_ms(20); // Stabilisierung
    while (!(PINB & (1 << PB0))); // Warte auf release
    brightness = brightness - 3;
    init_PWM();
}
// Begrenze die Helligkeit auf den Bereich von -3 bis 3
if (brightness > 3) {
    brightness = 3;
    } else if (brightness < -3) {
    brightness = 3;//wiederhole die Runde
}
```

in InitPWM()

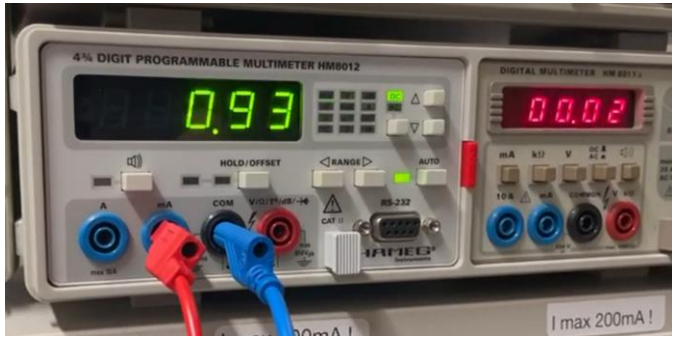
```
// Berechnung der PWM-Duty-Cycle-Werte basierend auf Helligkeitswert
OCR0A = 126 + (50 * brightness);
OCR0B = 126 - (50 * brightness); // OCR0B ist das Komplement von OCR0A
```

SLEEP MODUS POWERSAVE

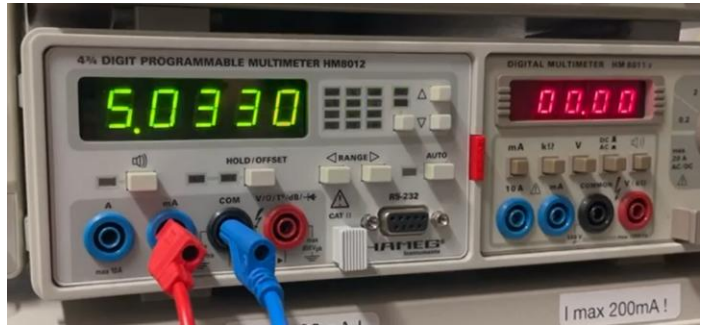
```
volatile uint8_t powerSaveMode = 0;
```
Globale Änderung

```
        if (!(PINB & (1 << PB1)) && !powerDownMode) {
            _delay_ms(20); // Stabilisierung
            while (!(PINB & (1 << PB1))); // Wait for button release
            powerSaveMode = !powerSaveMode;
        }
```
in ISR(PCINT0_vect)

```
while (1) {
    if (powerDownMode) {
        set_sleep_mode(SLEEP_MODE_PWR_DOWN);
        sleep_enable();
        sleep_cpu(); // Enter Power-Down Mode
    }
    else if (powerSaveMode) {
        set_sleep_mode(SLEEP_MODE_PWR_SAVE);
        sleep_enable();
        sleep_cpu(); // Enter Power-Save Mode
    }
    else {
        sleep_disable();
        init_PWM();
    }
}
```
in Main()



Power Save Mode



Normal Mode

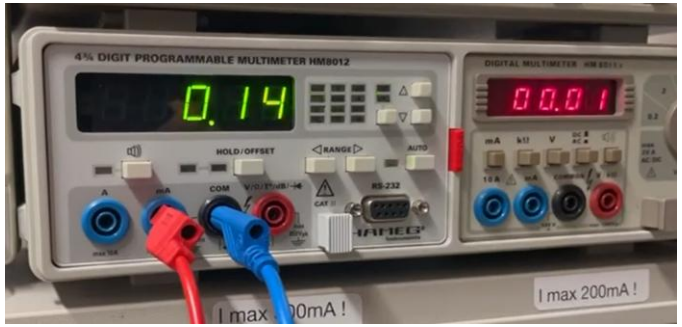
OFF/ON

Globale Änderung

```
volatile uint8_t powerDownMode = 0;
```

in ISR(PCINT0_vect)

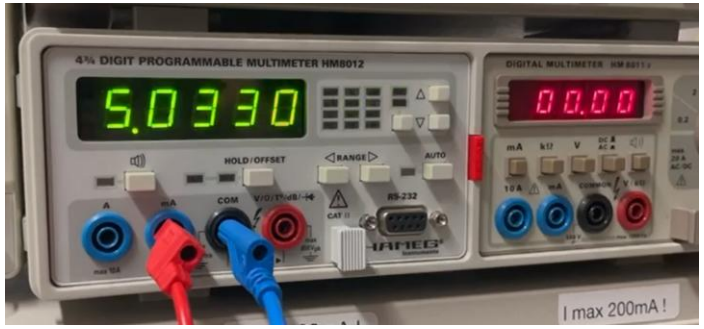
```
if (!(PINB & (1 << PB2))) {
    _delay_ms(20); // Stabilisierung
    while (!(PINB & (1 << PB2))); // Wait for button release
    powerDownMode = !powerDownMode;
    if (powerDownMode) {
        saveToEEPROM();
    }
}
```



Power Down Mode

Main()

```
while (1) {
    if (powerDownMode) {
        set_sleep_mode(SLEEP_MODE_PWR_DOWN);
        sleep_enable();
        sleep_cpu(); // Enter Power-Down Mode
    }
    else if (powerSaveMode) {
        set_sleep_mode(SLEEP_MODE_PWR_SAVE);
        sleep_enable();
        sleep_cpu(); // Enter Power-Save Mode
    }
    else {
        sleep_disable();
        init_PWM();
    }
}
```



Normal Mode

SEKUNDEN MODE

```
volatile uint8_t sec_Mode = 0;
```

Globale Änderung

```
if (!(PINB & ((1 << PB0) | (1 << PB1)))) {
    _delay_ms(20); // Stabilisierung
    while (!(PINB & ((1 << PB0) | (1 << PB1)))); // Wait for button release
    sec_Mode = !sec_Mode;
    init_PWM();
}
```

in ISR(PCINT0_vect)

```
void enable_LEDs() {
    PORTC = hours;

    if(sec_Mode){
        PORTD = seconds;
    }
    else{
        PORTD= minutes;
    }
}
```

in InitPWM()

UHR EINSTELLEN

in ISR(PCINT0_vect)

```
if (!(PINB & ((1 << PB1) | (1 << PB2)))) {
    _delay_ms(20); // Stabilisierung
    while (!(PINB & ((1 << PB1) | (1 << PB2)))); // Wait for button release
    ++minutes;
    if(minutes==60){
        minutes=0;
    }
    //init_PWM();
}
```

in ISR(PCINT0_vect)

```
if (!(PINB & ((1 << PB0) | (1 << PB2)))) {
    _delay_ms(20); // Stabilisierung
    while (!(PINB & ((1 << PB0) | (1 << PB2)))); // Wait for button release
    ++hours;
    if(hours==24){
        hours=0;
    }
    //init_PWM();
}
```

LETZER ZUSTAND SPEICHERN BEIM AUSMACHEN

```
int main(void) {
    // Aktualisiere die Werte im EEPROM
    loadFromEEPROM();

void saveToEEPROM() {
    eeprom_update_word((uint16_t*)EEPROM_SECONDS_ADDR, seconds);
    eeprom_update_word((uint16_t*)EEPROM_MINUTES_ADDR, minutes);
    eeprom_update_word((uint16_t*)EEPROM_HOURS_ADDR, hours);
//  eeprom_update_byte((int8_t*)EEPROM_BRIGHTNESS_ADDR, brightness);
}

void loadFromEEPROM() {
    seconds = eeprom_read_word((uint16_t*)EEPROM_SECONDS_ADDR) %60;
    minutes = eeprom_read_word((uint16_t*)EEPROM_MINUTES_ADDR)%60;
    hours = eeprom_read_word((uint16_t*)EEPROM_HOURS_ADDR)%24;
//  brightness = eeprom_read_byte((int8_t*)EEPROM_BRIGHTNESS_ADDR);
}
```

```
if (!(PINB & (1 << PB2))) {
    _delay_ms(20); // Stabilisierung
    while (!(PINB & (1 << PB2))); // W
    powerDownMode = !powerDownMode;
    if (powerDownMode) {
        saveToEEPROM();
    }
}
```

DANK FÜR IHRE AUFMERKSAMKEIT