Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Laborbericht 2

Timer

Ausarbeitung Gruppe 20

Laborbericht 2 eingereicht von
Klärmann, Nico
Matrikelnummer: 2603380
Laborgruppe 20

im Rahmen der Vorlesung *INF3 S22*
im Studiengang Media Systems
am Department Medientechnik der Fakultät DMI
an der Hochschule für Angewandte Wissenschaften Hamburg

Lehrende*r: Prof. Dr. Tessa Taefi

Eingereicht am: 03.05.2022

Zusammenfassung

In der zweiten Laborübung des Fachs „Informatik und Elektronik“ geht es um das Verständnis um die Timermodule des Atmel „Atmega 328P“. Die Timermodule werden genutzt, um die LEDs eines selbstentwickelten Shields anzusteuern und einen auf dem Shield verbauten Button zu entprellen.

Vor der Anwendung auf dem Board werden die Programme mittels Blockschaltbildern und UML Diagrammen geplant, und in der Entwicklungsumgebung „Microchip Studio“, sowie der Simulationsumgebung „SimulIDE“ getestet.

Die Ergebnisse und Erkenntnisse wurden im Anschluss wissenschaftlich Dokumentiert.

Inhaltsverzeichnis

ABKÜRZUNGSVERZEICHNIS

MCS	Microchip Studio
MC	Micro Controller
UML	Unified Modeling Language
ISR	Interrupt Service Routine
IO	Input / Output

ABBILDUNGSVERZEICHNIS

1 LABOR 2 - TIMER

Dieser Laborbericht entsteht als Teil einer Prüfungsleistung im Fach “Informatik und Elektronik”, in dessen Rahmen die Aufgaben des Labor 2 bearbeitet, dokumentiert und geprüft werden sollen. Die Abnahme erfolgt über eine praktische Vorstellung der Ergebnisse während einer Laboreinheit.

Geprüft werden das Verständnis und der Umgang mit den Timermodulen des Atmel 328P.

Dabei sollen der Atmel 328 und das selbstentwickelte LED Shield in Betrieb genommen werden. Selbst geschriebene, den Anforderungen der Aufgaben entsprechende Firmware soll die Timermodule des Controllers zur Ein- und Ausgabe konfigurieren und nutzen, um einen Pushbutton zu entprellen.

1.1 ZIELE DER LABORAUFGABEN

Durch das Ausführen der selbstgeschriebenen Firmware auf dem ATmega 328P und dem Shield sollen die, in der Vorlesung erlernten Kenntnisse demonstriert werden.

Dabei sollen Programme geschrieben werden, die verschiedene Funktionen des Boards nutzen und auftretende Probleme analysiert und behoben werden.

1.2 ANFORDERUNGSANALYSE

Das Labor 2 stellt folgende Anforderungen:

Aufgabe 1: Lauflicht

Es soll ein C Programm entwickelt werden, dass die 10 LEDs des Shields ansteuert.

Timer sollen genutzt werden, um eine LED für ca. eine Sekunde leuchten zu lassen, diese abzuschalten und die benachbarte LED aufleuchten zu lassen. Sind alle zehn LEDs durchlaufen worden, soll wieder bei der ersten LED angefangen werden.

Spezifisch sollen für diese Aufgaben die MC internen Timer-Register anstelle der im „util/delay.h“ Header verfügbaren „_dealy“ Funktion genutzt werden.

Aufgabe 2: Lauflicht und Taster

In dieser Aufgabe soll das Lauflicht aus Aufgabe 1 weiterverwendet werden.

In diesem Fall soll das Lauflicht nicht durch einen Timer, sondern durch einen Button weitergeschaltet werden. Der Timer soll stattdessen zum Entprellen des Buttons dienen, damit immer nur genau eine LED weitergeschaltet wird.

Des Weiteren soll keine Lauflicht entstehen, wenn der Button gedrückt gehalten wird.

2 LABORVORBEREITUNG UND METHODIK

Sämtliche Aufgaben wurden unter MS Windows 10 und teilweise über ein durch USB verbundenes Atmel 328P Board ausgeführt, dazu wurden folgende Software und Datensätzen genutzt.

Blockschaltbilder und Ablaufdiagramme wurden in MS Visio erstellt.

Software wurde anhand vorher erstellter Ablaufdiagramme in MCS entwickelt, welches ebenfalls als Hauptwerkzug zum Testen und Debugging verwendet wurde.

Weitere Tests und Erstellung von Schaltplänen erfolge in SimulIDE

MCS wurde aufgrund der tiefen Einbindung aller Atmel Produkte gewählt, da so einfaches Simulieren der Software sowie Hardware-Debugging direkt aus der IDE möglich war.

SimulIDE bietet eine Alternative Firmware zu testen sofern kein Chip zur Hand ist und ermöglicht aufgrund der Vorkonfiguration verschiedener Microcontroller einfaches Prototyping von Schaltungen und eine schnelle Erstellung von Schaltplänen.

Die Ausführung und Vorbereitung des Labors wurden durch die Verfügbarkeit von Microcontrollern außerhalb der Laborzeiten limitiert, was hauptsächlich durch Simulation in SimulIDE ergänzt werden konnte.

3 DOKUMENTATION DER LABORARBEIT

Im Folgenden wird die Planung und Durchführung der „Laboreinheit 2: Timer“ dokumentiert.
Dies wird analog zu den von Prof. Dr. Taefi zur Verfügung gestellten Folien in drei Unterpunkte gegliedert.
Des Weiteren folgen jeweils Planung und Vorbereitung des Labors separat von Implementierung und Beobachtung.

3.1 PLANUNG DER LABORAUFGABEN

Sämtliche Laboraufgaben wurden im Vorhinein durch Material aus der Vorlesung „Informatik und Elektronik“ und Methoden aus Kapitel 2 vorbereitet und konnten erst zum Labortermin getestet und abgenommen werden.

3.1.1 Lauflicht

Laboraufgabe 1 erfordert das entwickeln eines Programms, dass durch Nutzung der Timermodule des Controllers die LEDs des Shields nacheinander im Sekundentakt an und aus schaltet. Nach einem beendeten Durchlauf soll dieser wiederholt werden.

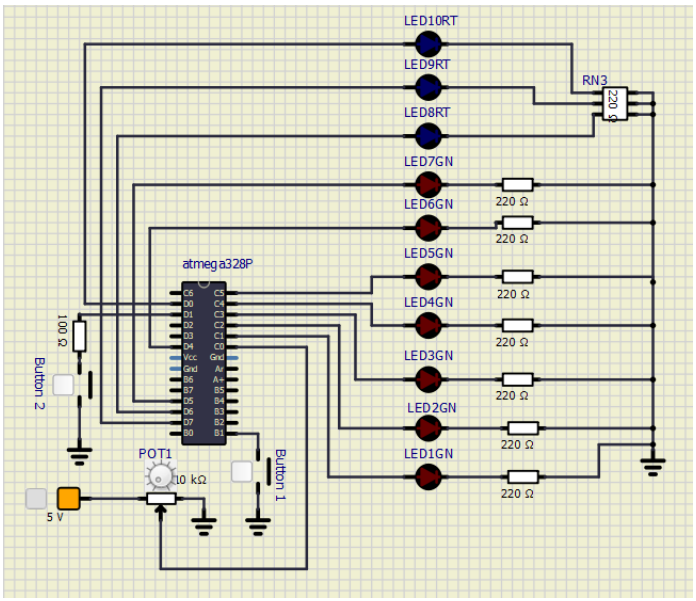

Abbildung 1: Schaltplan Atmega 328P mit selbstentwickeltem LED Shield

Das Setup von Controller mit LED Shield kann hierbei durch das Blockschaltbild in Abbildung 2 beschrieben werden. Hierbei werden über die Outputs des Controllers die LEDs des Shields der Konfiguration entsprechend angesteuert.

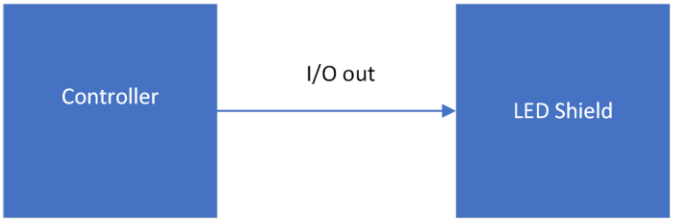

Abbildung 2: Blockschaltbild TimerCompare_Blink

Die Funktionalität des Programms wird im Ablaufplan in Abbildung 3 beschrieben,
welcher zeigt, dass nach der Initialisierung aller Notwendigen Register eine endlose Schleife gestartet wird. Das Inkrementieren der Laufvariable „Counter“, sowie die Steuerung der LEDs wird durch die „Timer Compare Interrupt“ Service Routine gehandhabt, welche mit jedem Output Vergleich des Timers aufgerufen wird.
Dieser wird alle 155 Ticks aufgerufen, was bei 16MHz ca 10 Millisekunden entspricht.
Somit können für die Kontrolle der LEDs einsekündige Zeit-Abstände gemessen werden.

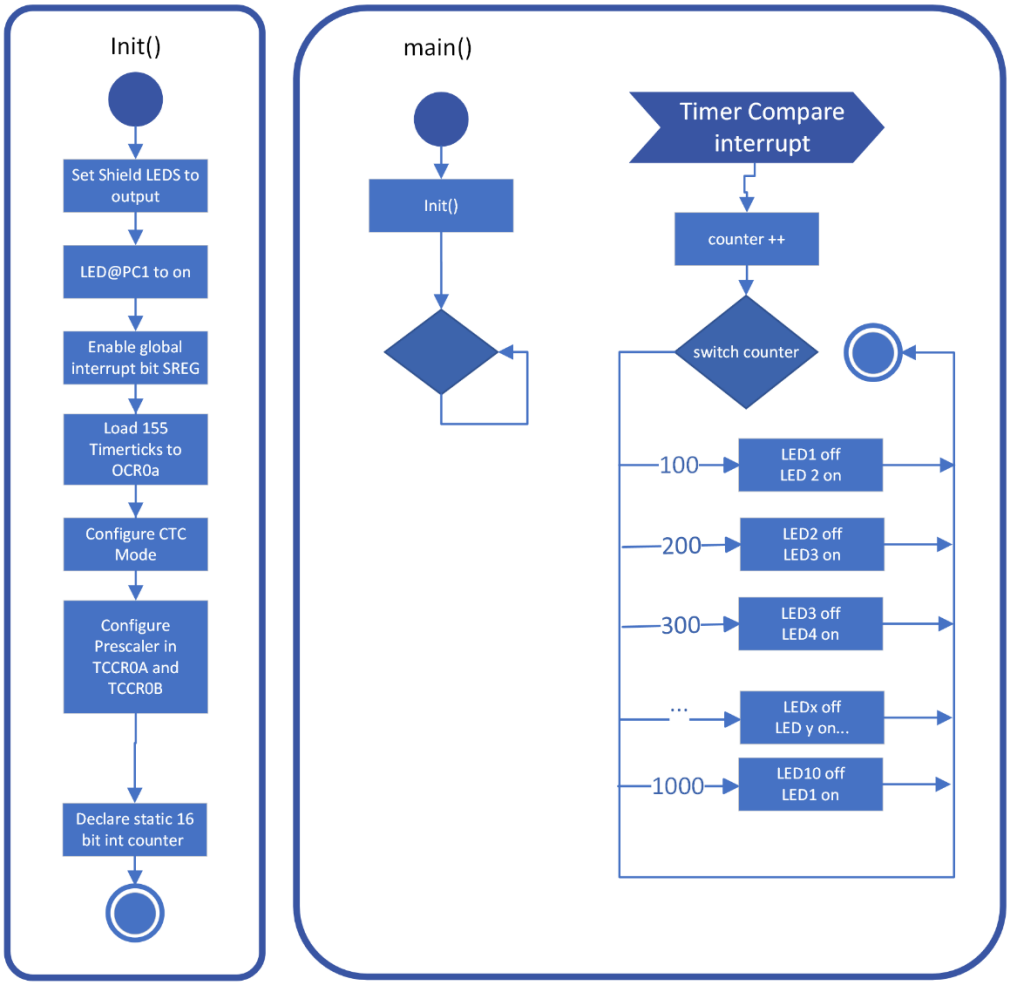

Abbildung 3: UML Aktivitätsdiagram TimerCompare_Blink

3.1.2 Lauflicht und Taster

In *Laboraufgabe 2* soll das Setup aus *Laboraufgabe 1* weiterverwendet werden. Die Aufgabe ist es, das Programm so anzupassen, dass die LED nicht durch einen Timer Compare Interrupt weitergeschaltet werden, sondern durch eine Benutzereingabe über einen Button.

Die Aufgabe des Timers soll es sein, den Button zu entprellen, d.h. dafür zu sorgen, dass der Button beim einmaligen Drücken nur eine LED weiterschaltet, und beim Gedrückthalten kein Lauflicht entsteht.

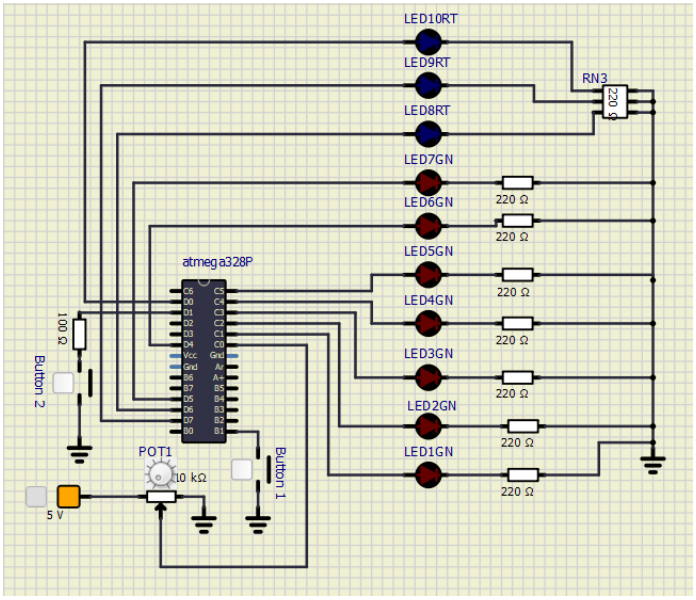

Abbildung 4: Schaltplan Atmega 328P mit selbstentwickeltem LED Shield

Für dieses Programm wird dem Blockschaltbild aus Aufgabe 1 ein Taster hinzugefügt. Dieser ist bereits auf dem Shield verbaut und wird durch die Änderung der In- und Output Ports aktiviert.

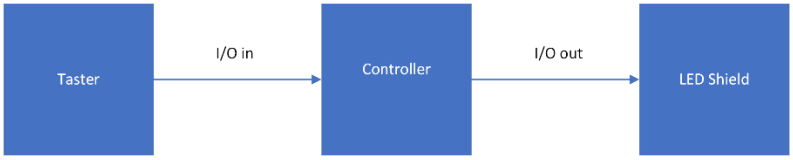

Abbildung 5: Blockschaltbild Atmega 328P mit Shield

Für die Realisierung wurde der Ablauf des Interrupts in eine eigene Funktion ausgelagert. Die Schleife in der „Main“ Methode wurde um eine Button-Abfrage erweitert, welche prüft, ob eine Nutzereingabe getätigt wurde und diese dann mit dem vorherigen Status des Buttons abgleicht. So wird der Wert „ButtonStatus“, welcher den Status der Nutzereingabe repräsentiert, nur dann im Interrupt erfasst, wenn er während des letzten Zyklus 0, also nicht gedrückt war.

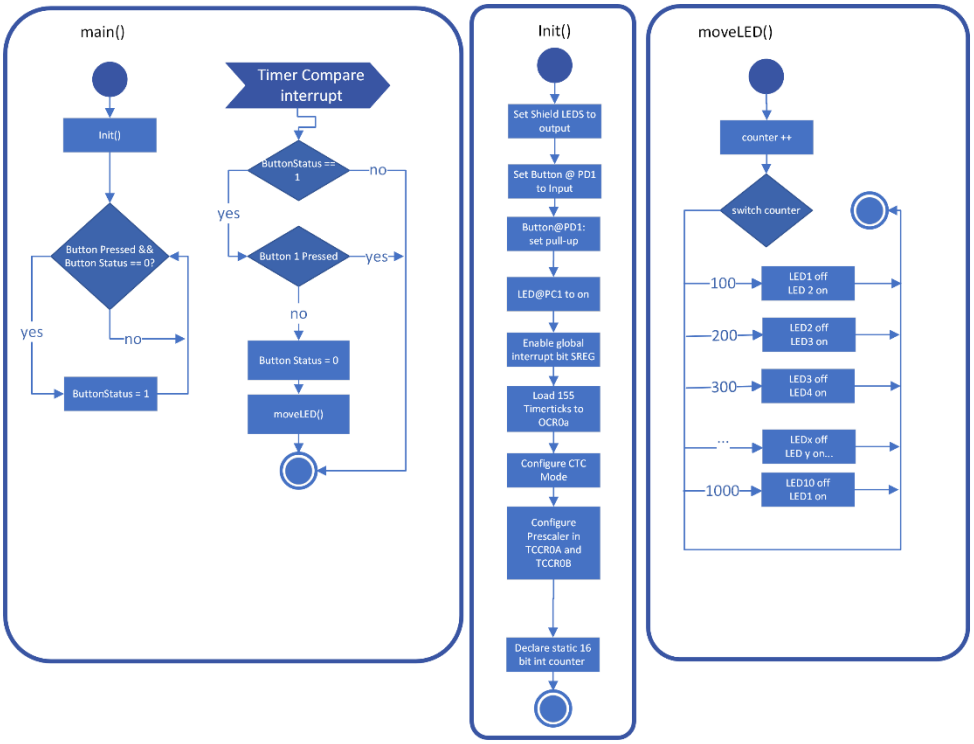

Abbildung 6: UML Aktivitätsdiagram LEDNext

Die Schleife im Interrupt fragt ebenfalls den Status des Buttons ab und ruft dem Status hier nur dann moveLED auf, wenn die Nutzereingabe nicht mehr getätigt wird. In diesem Falle reagiert das Programm also erst beim Loslassen des Buttons und gibt dabei auch den Wert „ButtonStatus“ wieder frei.

3.2 IMPLEMENTATION UND TESTUNG

Implementation des zuvor geplanten Codes, sowie Tests aus Aufgaben wurden zum Labortermin mithilfe der Planung aus Kapitel 3.1 ausgeführt und anschließend dokumentiert.

3.2.1 Laden und Ausführen von „TimerCompare_Blink“ (Aufgabe 1)

Die Software konnte aufgrund von Kompatibilitätsproblemen bezüglich der Tick-Rate des simulierten Chips nicht vollständigen in SimulIDE getestet werden, und wurde nach Tests in MCSs IO Debugger auf den Micro Controller geflasht.

Abbildung 7 und 8 zeigen Ausschnitte der ISR, welche die LEDs steuert.

```
ISR(TIMER0_COMPA_vect){

    static volatile uint16_t oneSecondEqual = 60;
    static volatile uint16_t counter = 0;
    counter ++;
    switch (counter){
                                        // 0ms = C1 = 1GN ON
        case 100:                       // 100 * 10ms = 1 second passed = 2GN on
            PORTC       &= ~(1<<1);     // 1GN off
            PORTC       |= (1<<2);      // 2GN on
            break;
        case 200:                       // 200 * 10ms = 2 seconds passed = 3GN on
            PORTC       &= ~(1<<2);     // 2GN off
            PORTC       |= (1<<3);      // 3GN on
            break;
        case 300:                       // 300 * 10ms = 3 seconds passed = ....
            PORTC       &= ~(1<<3);     // 3GN off
            PORTC       |= (1<<4);      // 4GN on
            break;
```

*Abbildung 7: TimerCompare_Blink ISR Beispiel bis 300*10ms*

Wie in Abbildung 7 zu sehen, werden einsekündige Intervalle erreicht, indem der in der „init“ Methode Gesetzte „Timer-Reset Intervall“ von 10ms in Schritten von 100 multipliziert wird. Somit agiert der Gezeigte Switch dann mit den entsprechenden LEDs, um ein „Lauflicht“ zu erzeugen.

Sobald die Laufvariable „Counter“ den Wert 1000 erreicht, wird sie auf 0 zurückgesetzt und schaltet das erste LED erneut an, um einen neuen Durchlauf zu starten.

```
        case 900:
            PORTD       &= ~(1<<7);     // 9RT off
            PORTB       |= 0b00000001;  //|= (1<<0); // 10RT on
            break;
        case 1000:
            PORTB       &= ~0b00000001; //&= ~(1<<0); // 10RT off
            PORTC       |= (1<<1);      // 1GN on
            counter = 0;
            break;
    }
}
```

Abbildung 8: TimerCompare_Blink Ausschnitt der ISR 2, welches den „Reset“ Fall zeigt

3.2.2 Hinzufügen des Tasters (Aufgabe 2)

Abbildung 9 ATmega 328P mit LED Shield

Das Lauflicht, ist nun durch den Taster zu steuern. Dazu muss die Nutzerneingabe konstant abgefragt werden. So wird der Wert „Counter“ nur durch eine vollendete Nutzereingabe (Drücken und Loslassen) inkrementiert. Mithilfe eine „Timer-Compare“ Interupts, wird regelmäßig der Wert „Buttonstatus“ mit dem aktuellen Status der Nutzereingabe verglichen.
Sobald hierdurch sichergestellt wurde, dass eine Eingabe getätigt und vollendet wurde, wird „ButtonStatus“ zurückgesetzt, die Laufvariable „Counter“ inkrementiert und die damit verbundene Methode „switchLED“ aufgerufen.

Abbildung 10 und 11 zeigen die Unterschiede des Quellcodes im Vergleich zu „TimerCompare_Blink“.

```
static volatile uint8_t buttonStatus = 0; // 0 = PIND1 == 1

ISR(TIMER0_COMPA_vect){
    if(buttonStatus == 1 && !BUTTON_ONE_PRESS) {
        buttonStatus = 0;
        counter += 100;
        switchLED();
    }
}
```

Abbildung 10: _NextLED Timer Compare SR

```
int main(void){

    init();

    while (1){
        if(BUTTON_ONE_PRESS && buttonStatus == 0){
            buttonStatus = 1;
        }
    }
}
```

Abbildung 11: _NextLED main Methode

Ebenfalls wurde die Schleife zum Ansteuern der LEDs aus Aufgabe 1 (Abbildung 7 und 8) als eigenständige Methode „switchLED“ aus der ISR ausgelagert.

4 FAZIT

Während der Vorbereitung und der Durchführung des Labors konnten mit den verschiedenen Timermodi experimentiert werden. Es sind verschiedene Anwendungsfälle klar geworden, und wie man sie zur Lösung von Problemen aus dem ersten Labor einsetzen kann.

Die Erkenntnisse dieser Laborübung tragen zu einer besseren Entscheidungsfindung bei, wenn es um die Auswahl von Timereinstellungen für verschiedene Zwecke geht, und wann der Einsatz eines Timers sinnvoll oder effizienter ist als andere Methoden.

ANHANG

Der Quellcode zu “TimerCompare_Blink” (Aufgabe 1) und “_NextLED” (Aufgabe 2) ist zusätzlich separat in C Dateien in der Abgabe zu finden.

```
#define F_CPU16E6
#include <avr/io.h>
#include <avr/interrupt.h>

#define LED_ON PORTC        |= (1<<1)
#define LED_OFF PORTC       &= ~(1<<1)
#define LED_TOGGLE PORTC    ^= (1<<1)

void init(void){
    // LED
    DDRB |= 0b00000001;
    DDRC |= 0b00111110; // PC1-6 as output
    DDRD |= 0b11110001; // PD0-4 as output

    PORTC |= (1<<1); // LED 1 one

    // Interrupt
    sei();
    TIMSK0 |= (1<<OCIE0A);          // Timer0 A Match enable
    OCR0A = 155;                    // reset compare 10ms

    TCCR0A |= (1<<WGM01);           // Configure CTC Mode
    TCCR0A &= ~(1<<WGM00);
    TCCR0B &= ~(1<<WGM02);

    TCCR0B |=(1<<CS02) | (1<<CS00); // Prescaler 1024
    TCCR0B &= ~(1<<CS01);
}
```

```
ISR(TIMER0_COMPA_vect){
    static volatile uint16_t oneSecondEqual = 60;
    static volatile uint16_t counter = 0;
    counter ++;
    switch (counter){
        // 0ms = C1 = 1GN ON
        case 100:                   // 100 * 10ms = 1 second passed = 2GN
on
            PORTC       &= ~(1<<1); // 1GN off
            PORTC       |= (1<<2);  // 2GN on
            break;
        case 200:                   // 200 * 10ms = 2 seconds passed = 3GN
on
            PORTC       &= ~(1<<2); // 2GN off
            PORTC       |= (1<<3);  // 3GN on
            break;
        case 300:                   // 300 * 10ms = 3 seconds passed = ....
            PORTC       &= ~(1<<3); // 3GN off
            PORTC       |= (1<<4);  // 4GN on
            break;
        case 400:                   // 400 * 10ms = 4......
            PORTC       &= ~(1<<4); // 4GN off
            PORTC       |= (1<<5);  // 5GN on
            break;
        case 500:                   // 500 * .......
            PORTC       &= ~(1<<5); // 5GN off
            PORTD       |= (1<<4);  // 6GN on
            break;
        case 600:
            PORTD       &= ~(1<<4); // 6GN off
            PORTD       |= (1<<5); // 7GN on
            break;
        case 700:
            PORTD       &= ~(1<<5); // 7GN off
            PORTD       |= (1<<6); // 8RT on
            break;
        case 800:
            PORTD       &= ~(1<<6); // 8RT off
            PORTD       |= (1<<7); // 9RT on
            break;
```

```
        case 900:
```

```
            PORTD       &= ~(1<<7); // 9RT off
            PORTB       |= 0b00000001;//|= (1<<0); // 10RT on
            break;
        case 1000:
            PORTB       &= ~0b00000001;//&= ~(1<<0); // 10RT off
            PORTC       |= (1<<1); // 1GN on
            counter = 0;
            break;
    }
}

int main(void){
    init();
    while (1){}
}
```

Quellcode _NextLED

```
#define F_CPU 16E6
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define BUTTON_ONE_PRESS (!(PIND & (1<<1)))

static volatile uint16_t counter = 0;
```

```
void init(void){
    // LED
    DDRB |= 0b00000001; // PB0 as output
    DDRC |= 0b00111110; // PC1-6 as output
    DDRD |= 0b11110001; // PD0-4 as output

    PORTC   |= (1<<1);  // LED 1 one

    // Button
    DDRD &= ~(1<<1);    //Configure PD1 as Input
    PORTD |= 1<<1;      //Enable Internal Pull-Up at PD1


    // Interrupt
    sei();
    TIMSK0 |= (1 << OCIE0A);    // Timer0 A Match enable
    OCR0A = 155;                // reset compare 10ms

    TCCR0A |= (1 << WGM01);     // Configure CTC Mode
    TCCR0A &= ~(1 << WGM00);
    TCCR0B &= ~(1 << WGM02);

    TCCR0B |= (1 << CS02) | (1 << CS00); // Prescaler 1024
    TCCR0B &= ~(1 << CS01);
}
```

```
static volatile uint8_t buttonStatus = 0; // 0 = PIND1 == 1
```

```
ISR(TIMER0_COMPA_vect){
    if(buttonStatus == 1 && !BUTTON_ONE_PRESS) {
        buttonStatus = 0;
        counter += 100;
        switchLED();
    }
}
```

```
void switchLED(){
    //static volatile uint16_t oneSecondEqual = 60;
    switch (counter){
        case 100:
        PORTC       &= ~(1<<1); // 1GN off
        PORTC       |= (1<<2); // 2GN on
        break;
        case 200:
        PORTC       &= ~(1<<2); // 2GN off
        PORTC       |= (1<<3); // 3GN on
        break;
        case 300:
        PORTC       &= ~(1<<3); // 3GN off
        PORTC       |= (1<<4); // 4GN on
        break;
        case 400:
        PORTC       &= ~(1<<4); // 4GN off
        PORTC       |= (1<<5); // 5GN on
        break;
        case 500:
        PORTC       &= ~(1<<5); // 5GN off
        PORTD       |= (1<<4); // 6GN on
        break;
        case 600:
        PORTD       &= ~(1<<4); // 6GN off
        PORTD       |= (1<<5); // 7GN on
        break;
        case 700:
        PORTD       &= ~(1<<5); // 7GN off
        PORTD       |= (1<<6); // 8RT on
        break;
        case 800:
        PORTD       &= ~(1<<6); // 8RT off
        PORTD       |= (1<<7); // 9RT on
        break;
        case 900:
        PORTD       &= ~(1<<7); // 9RT off
        PORTB       |= (1<<0); // 10RT on
        break;
        case 1000:
        PORTB       &= ~(1<<0); // 10RT off
        PORTC       |= (1<<1); // 1GN on
        counter = 0;
        break;
    }
}
```

```
int main(void){
    init();

    while (1){
        if(BUTTON_ONE_PRESS && buttonStatus == 0){
            buttonStatus = 1;
        }
    }
}
```

EIGENSTÄNDIGKEITSERKLÄRUNG

Hiermit versichern wir, dass wir das vorliegende Dokument selbständig und nur mit den angegebenen Hilfsmitteln verfasst haben. Alle Passagen, die wir wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen haben, haben wir deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

	Sebastian Kühne
03.05.2022	Nico Klärmann
Datum	Unterschrift