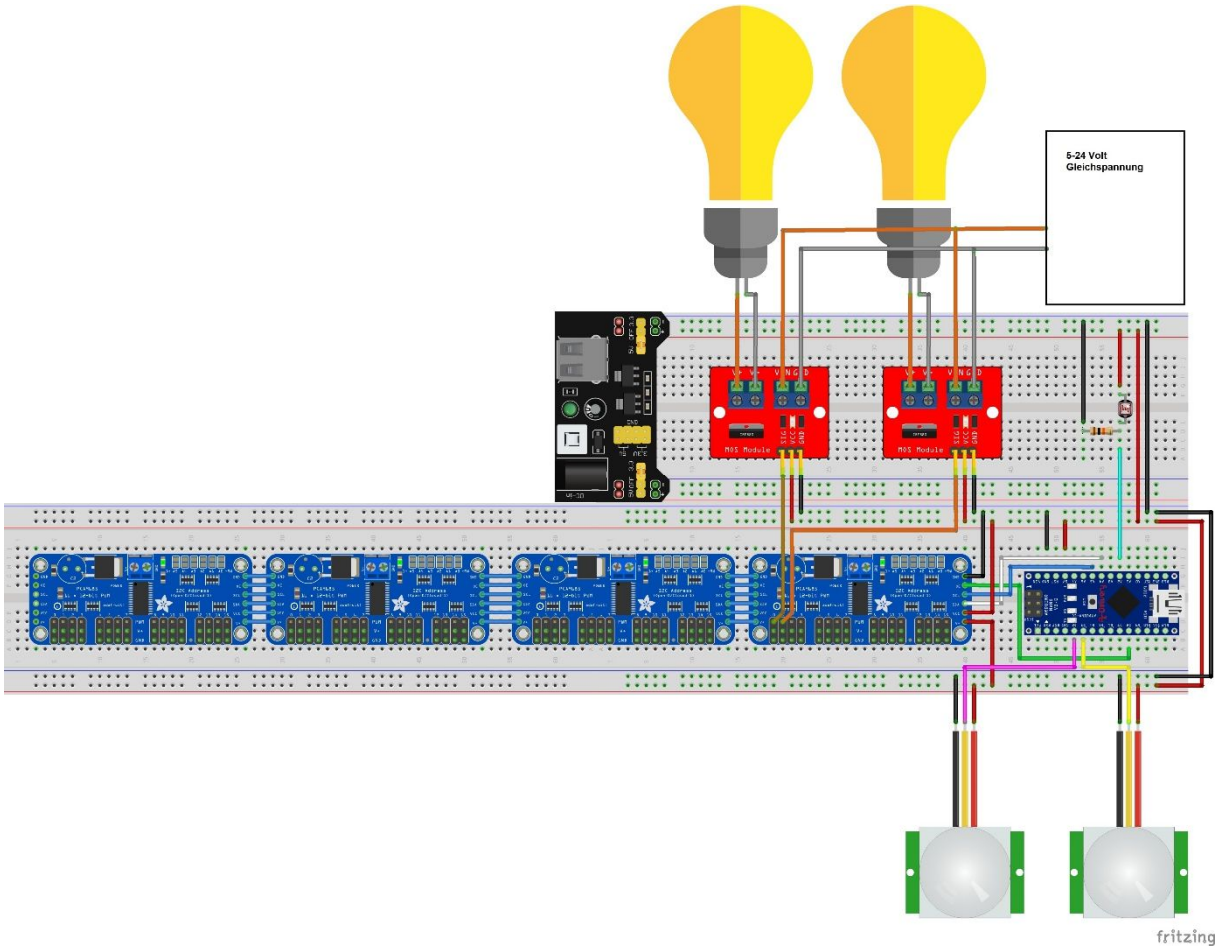
Eine elegante automatische Treppenbeleuchtung (Teil5)

Hallo und Willkommen zu dem letzten der Teil der Reihe „elegante automatische Treppenbeleuchtung“. Heute runden wir unsere Steuerung um eine komfortable Konfigurationsmöglichkeit aller Betriebsparameter ab. Alle Betriebsparameter können nun über die serielle Schnittstelle im Ruhezustand (alle Treppenlichter inaktiv) bequem eingestellt werden und werden im internen EEPROM des uC’s abgespeichert. Somit bleiben alle Einstellungen auch bei einem Neustart oder bei Spannungsausfall erhalten. Alle einstellbaren Parameter werden am Ende des Dokumentes einzeln erklärt. Da die Funktion komplett in Software realisiert ist ändert sich der technische Aufbau um Vergleich zu Teil 4 der Reihe nicht. Trotzdem soll dieser vollständigkeitshalber nochmals dargestellt werden:



Auch die Teile Liste des Projektes und alle Hinweise aus den vorherigen Teilen ändern sich nicht:

Anzahl	Beschreibung	Anmerkung
2	PIR Modul HC-SR501 PIR	Bewegungssensor
bis 62	PCA9685 16 Kanal 12 Bit PWM Driver	Anzahl je nach Treppenzahl /16
1	Nano V3	
1	MB102 Netzteil Adapter	Für Breadboardaufbau
bis 992	IRF520 MOS Driver Modul 0-24V 5A	Anzahl je nach Treppenzahl
1	Netzteil für LED/Lampen für die Stufen	Maximal 24 Volt
1	10 KOhm Wiederstand	
1	LDR	Fotowiederstand

Es kann OHNE vorherige Anpassung folgender Code auf den Arduino hochgeladen werden:

```
#include <Wire.h>
#include <EEPROM.h>

#define PWM_Module_Base_Addr 0x40 // 10000000b  Das letzte Bit des Adressbytes definiert die auszuführende Operation. Bei Einstellung auf logisch 1  0x41 Modul 2 etc.. Adressbereich0x40 - 0x47
                                  // wird ein Lesevorgang auswählt, während eine logische 0 eine Schreiboperation auswählt.
#define OE_Pin  8               // Pin für Output Enable
#define CPU_LED_Pin 13          // Interne Board LED an Pin 13 (zu Debuggingzwecken)
#define PIRA_Pin 2
#define PIRB_Pin 3
#define Num_Stages_per_Module 16
#define LDR_Pin A2              // Analog Pin, über den die Helligkeit gemessen werden soll. (LDR Wiederstand)
// #define DEBUG
#define L_Sens_Scope 50
#define MaxInputBufferSize 5 // maximal 255 Zeichen anpassen an vlcdr


struct WiFiEEPromData
 {
  // Anpassbare Betriebsparameter (Konstanten)
  int Delay_ON_to_OFF = 10;         // Minimum Wartezeit bis zur "Aus Sequenz" in Sekunden
  int Overall_Stages =  8;        // maximale Stufenanzahl: 62 x 16 = 992
  int delay_per_Stage_in_ms = 100;
  int DayLight_Brightness_Border = 600; // Helligkeitsgrenze Automatik - Höherer Wert - Höhere Helligkeit
  byte Delay_Stages_ON = 20;
  byte Delay_Stages_OFF = 20;
  char ConfigValid[3]; //If Config is Vaild, Tag "TK" is required"
```

```
  };


// Globale Variablen
int Pwm_Channel = 0;
int Pwm_Channel_Brightness = 0;
bool Motion_Trigger_Down_to_Up = false;
bool Motion_Trigger_Up_to_Down = false;
bool On_Delay = false;
bool DayLight_Status = true;
bool DLightCntrl = true;
byte PWMModules = 0;
byte StagesLeft = 0;
// interrupt Control
volatile byte A60telSeconds24 = 0;
volatile byte Seconds24;
//Serial Input Handling
char TBuffer;
char Cbuffer[MaxInputBufferSize+1];      //USB Code Input Buffer
String Sbuffer = "";                     //USB String Input Buffer
int value;                           //USB Nummeric Input Buffer
byte Ccount { 0 };                       //Number received Chars
byte Inptype = 0;
boolean StrInput = false;
boolean NumberInput = false;
boolean DataInput = false;
boolean EnterInput = false;
byte MenueSelection = 0;
byte MnuState = 0;          // Maximale Menuetiefe 255 icl Sub
WiFiEEPromData MyConfig;


ISR(TIMER1_COMPA_vect)
{
A60telSeconds24++;
  if (A60telSeconds24 > 59)
    {
     A60telSeconds24 = 0;
     Seconds24++;
     if (Seconds24 > 150)
       {
         Seconds24 = 0;
       }
    }
}

void ISR_PIR_A()
{
bool PinState = digitalRead(PIRA_Pin);
if (PinState)
  {
  if (!(Motion_Trigger_Up_to_Down) and !(Motion_Trigger_Down_to_Up))
    {
```

```
   digitalWrite(CPU_LED_Pin,HIGH);
   Motion_Trigger_Down_to_Up = true;
   } // PIR A ausgelöst
  } else
  {
   digitalWrite(CPU_LED_Pin,LOW);
  }
}

void ISR_PIR_B()
{
bool PinState = digitalRead(PIRB_Pin);
if (PinState)
  {
   if (!(Motion_Trigger_Down_to_Up) and !(Motion_Trigger_Up_to_Down))
     {
     digitalWrite(CPU_LED_Pin,HIGH);
     Motion_Trigger_Up_to_Down = true;
     } // PIR B ausgelöst
  } else
  {
   digitalWrite(CPU_LED_Pin,LOW);
  }
}

void Init_PWM_Module(byte PWM_ModuleAddr)
{
  digitalWrite(OE_Pin,HIGH); // Active LOW-Ausgangsaktivierungs-Pin (OE).
  Wire.beginTransmission(PWM_ModuleAddr); // Datentransfer initiieren
  Wire.write(0x00);                  //
  Wire.write(0x06);                  // Software Reset
  Wire.endTransmission();              // Stoppe Kommunikation - Sende Stop Bit
  delay(400);
  Wire.beginTransmission(PWM_ModuleAddr); // Datentransfer initiieren
  Wire.write(0x01);                  // Wähle  Mode 2 Register (Command Register)
  Wire.write(0x04);                  // Konfiguriere Chip: 0x04:  totem pole Ausgang 0x00: Open drain
Ausgang.
  Wire.endTransmission();              // Stoppe Kommunikation - Sende Stop Bit
  Wire.beginTransmission(PWM_ModuleAddr); // Datentransfer initiieren
  Wire.write(0x00);                 // Wähle Mode 1 Register (Command Register)
  Wire.write(0x10);                 // Konfiguriere SleepMode
  Wire.endTransmission();             // Stoppe Kommunikation - Sende Stop Bit
  Wire.beginTransmission(PWM_ModuleAddr); // Datentransfer initiieren
  Wire.write(0xFE);                 // Wähle PRE_SCALE register (Command Register)
  Wire.write(0x03);                  // Set Prescaler. Die maximale PWM Frequent ist 1526 Hz wenn
das PRE_SCALEer Regsiter auf "0x03h" gesetzt wird. Standard : 200 Hz
  Wire.endTransmission();              // Stoppe Kommunikation - Sende Stop Bit
  Wire.beginTransmission(PWM_ModuleAddr); // Datentransfer initiieren
  Wire.write(0x00);                  // Wähle Mode 1 Register (Command Register)
  Wire.write(0xA1);                  // Konfiguriere Chip:  ERrlaube All Call I2C Adressen, verwende
interne Uhr,                          // Erlaube Auto Increment Feature
  Wire.endTransmission();             // Stoppe Kommunikation - Sende Stop Bit
}
```

```
void Init_PWM_Outputs(byte PWM_ModuleAddr)
{
  digitalWrite(OE_Pin,HIGH); // Active LOW-Ausgangsaktivierungs-Pin (OE).
   for ( int z = 0;z < 16 + 1;z++)
    {
     Wire.beginTransmission(PWM_ModuleAddr);
     Wire.write(z * 4 +6);      // Wähle PWM_Channel_ON_L register
     Wire.write(0x00);                  // Wert für o.g. Register
     Wire.endTransmission();
     Wire.beginTransmission(PWM_ModuleAddr);
     Wire.write(z * 4 +7);      // Wähle PWM_Channel_ON_H register
     Wire.write(0x00);                  // Wert für o.g. Register
     Wire.endTransmission();
     Wire.beginTransmission(PWM_ModuleAddr);
     Wire.write(z * 4 +8);    // Wähle PWM_Channel_OFF_L register
     Wire.write(0x00);       // Wert für o.g. Register
     Wire.endTransmission();
     Wire.beginTransmission(PWM_ModuleAddr);
     Wire.write(z * 4 +9);   // Wähle PWM_Channel_OFF_H register
     Wire.write(0x00);           // Wert für o.g. Register
     Wire.endTransmission();
    }
digitalWrite(OE_Pin,LOW); // Active LOW-Ausgangsaktivierungs-Pin (OE).
}

void setup()
{
   //Initalisierung
   Serial.begin(9600);
   pinMode(PIRA_Pin,INPUT);
   pinMode(PIRB_Pin,INPUT);
   pinMode(OE_Pin,OUTPUT);
   pinMode(CPU_LED_Pin,OUTPUT);
   pinMode(LDR_Pin,INPUT);
   PWMModules = MyConfig.Overall_Stages / 16;
   StagesLeft = ( MyConfig.Overall_Stages % 16) -1;
   if (StagesLeft >= 1) {PWMModules++;}
   Wire.begin(); // Initalisiere I2C Bus A4 (SDA), A5 (SCL)
   for (byte ModuleCount=0;ModuleCount < PWMModules;ModuleCount++)
    {
     Init_PWM_Module(PWM_Module_Base_Addr + ModuleCount);
     Init_PWM_Outputs(PWM_Module_Base_Addr + ModuleCount);
    }

   if (!(loadEEPROM_Config())) // Load Seetings from EEPROM
    {
    Serial.println(F("EEPROM Standard Settings saved."));
      MyConfig.Delay_ON_to_OFF = 10;         // Minimum Wartezeit bis zur "Aus Sequenz" in
Sekunden
      MyConfig.Overall_Stages =  8;        // maximale Stufenanzahl: 62 x 16 = 992
      MyConfig.delay_per_Stage_in_ms = 100;
```

```
    MyConfig.DayLight_Brightness_Border = 600; // Helligkeitsgrenze Automatik - Höherer Wert - Höhere Helligkeit
    MyConfig.Delay_Stages_ON = 20;
    saveEEPROM_Config();
   }
  noInterrupts();
  attachInterrupt(0, ISR_PIR_A, CHANGE);
  attachInterrupt(1, ISR_PIR_B, CHANGE);
  TCCR1A = 0x00;
  TCCR1B = 0x02;
  TCNT1 = 0;     // Register mit 0 initialisieren
  OCR1A =  33353;     // Output Compare Register vorbelegen
  TIMSK1 |= (1 << OCIE1A);  // Timer Compare Interrupt aktivieren
  interrupts();
  Serial.println(F("Init_Complete"));
}

/** Save Config to EEPROM */

bool loadEEPROM_Config()
{
 bool RetValue;

 EEPROM.get(0, MyConfig);
 EEPROM.end();
 if (String(MyConfig.ConfigValid) = String("TK"))
  {
   RetValue = true;
  } else
  {
   RetValue = false; // Settings not found.
  }
  return RetValue;
}

/** Store Config to EEPROM */
bool saveEEPROM_Config()
{
  strncpy( MyConfig.ConfigValid , "TK", sizeof(MyConfig.ConfigValid) );
  EEPROM.put(0, MyConfig);
  EEPROM.end();
  return true;
}

bool DayLightStatus ()
{
int SensorValue = 0;
bool ReturnValue = true;
SensorValue = analogRead(LDR_Pin);
#ifdef DEBUG
Serial.print(F("DayLightStatus: "));
Serial.print(SensorValue);
#endif
```

```
if (SensorValue > MyConfig.DayLight_Brightness_Border)
 {
  if ((DayLight_Status) and (SensorValue > MyConfig.DayLight_Brightness_Border +
L_Sens_Scope))
    {
    ReturnValue = false;
    DayLight_Status = false;
    } else if (!(DayLight_Status))
    {
    ReturnValue = false;
    DayLight_Status = false;
    }
  #ifdef DEBUG
  Serial.println(F(" OFF"));
  #endif
 } else
 {
  if ((DayLight_Status) and (SensorValue > MyConfig.DayLight_Brightness_Border -
L_Sens_Scope))
    {
    ReturnValue = true;
    DayLight_Status = true;
    } else if (!(DayLight_Status))
    {
    ReturnValue = true;
    DayLight_Status = true;
    }
  #ifdef DEBUG
  Serial.println(F(" ON"));
  #endif
 }
return ReturnValue;
}

void Down_to_Up_ON()
{
#ifdef DEBUG
Serial.println(F("Down_to_Up_ON"));
#endif
byte Calc_Num_Stages_per_Module = Num_Stages_per_Module;
for (byte ModuleCount=0;ModuleCount < PWMModules;ModuleCount++)
        {
        Pwm_Channel = 0;
        Pwm_Channel_Brightness = 4095;
        if ((StagesLeft >= 1) and (ModuleCount == PWMModules -1))
                {
                Calc_Num_Stages_per_Module = StagesLeft;
                }
        else
                {
        Calc_Num_Stages_per_Module = Num_Stages_per_Module;
                }
        Pwm_Channel = 0;
```

```
            Pwm_Channel_Brightness = 0;
            while (Pwm_Channel < Calc_Num_Stages_per_Module +1)
              {
              Wire.beginTransmission( PWM_Module_Base_Addr + ModuleCount);
              Wire.write(Pwm_Channel * 4 +8);    // Wähle PWM_Channel_0_OFF_L register
              Wire.write((byte)Pwm_Channel_Brightness & 0xFF);       // Wert für o.g. Register
              Wire.endTransmission();
              Wire.beginTransmission( PWM_Module_Base_Addr + ModuleCount);
              Wire.write(Pwm_Channel * 4 +9);   // Wähle PWM_Channel_0_OFF_H register
              Wire.write((Pwm_Channel_Brightness >> 8));           // Wert für o.g. Register
              Wire.endTransmission();
              if (Pwm_Channel_Brightness < 4095)
               {
               Pwm_Channel_Brightness = Pwm_Channel_Brightness + MyConfig.Delay_Stages_ON;
               if (Pwm_Channel_Brightness > 4095) {Pwm_Channel_Brightness = 4095;}
               } else if ( Pwm_Channel < Num_Stages_per_Module +1)
               {
                 Pwm_Channel_Brightness = 0;
                      delay(MyConfig.delay_per_Stage_in_ms);
                 Pwm_Channel++;
               }
            }
   }
}

void Up_to_DOWN_ON()
{
#ifdef DEBUG
Serial.println(F("Up_to_DOWN_ON "));
#endif
byte Calc_Num_Stages_per_Module = Num_Stages_per_Module;
int ModuleCount = PWMModules - 1;
while (ModuleCount >= 0)
            {
            Pwm_Channel_Brightness = 0;
            if ((StagesLeft >= 1) and (ModuleCount == PWMModules -1))
                        {
                        Calc_Num_Stages_per_Module =  StagesLeft;

                        }
            else
                        {
                        Calc_Num_Stages_per_Module = Num_Stages_per_Module;
                        }
            Pwm_Channel = Calc_Num_Stages_per_Module;
            while (Pwm_Channel > -1)
              {
              Wire.beginTransmission( PWM_Module_Base_Addr + ModuleCount);
              Wire.write(Pwm_Channel * 4 +8);    // Wähle PWM_Channel_0_OFF_L register
              Wire.write((byte)Pwm_Channel_Brightness & 0xFF);       // Wert für o.g. Register
              Wire.endTransmission();
              Wire.beginTransmission(PWM_Module_Base_Addr + ModuleCount);
              Wire.write(Pwm_Channel * 4 +9);   // Wähle PWM_Channel_0_OFF_H register
```

```
              Wire.write((Pwm_Channel_Brightness >> 8));           // Wert für o.g. Register
              Wire.endTransmission();
              if (Pwm_Channel_Brightness < 4095)
               {
               Pwm_Channel_Brightness = Pwm_Channel_Brightness + MyConfig.Delay_Stages_ON;
               if (Pwm_Channel_Brightness > 4095) {Pwm_Channel_Brightness = 4095;}
               } else if ( Pwm_Channel >= 0)
              {
                 Pwm_Channel_Brightness = 0;
               delay(MyConfig.delay_per_Stage_in_ms);
                Pwm_Channel--;
                 if ( Pwm_Channel < 0)
                 {
                   Pwm_Channel =0;
                   break;
                 }
               }
              }
            ModuleCount = ModuleCount -1;
            }
}

void Down_to_Up_OFF()
{
#ifdef DEBUG
Serial.println(F("Down_to_Up_OFF"));
#endif
byte Calc_Num_Stages_per_Module = Num_Stages_per_Module;
for (byte ModuleCount=0;ModuleCount < PWMModules;ModuleCount++)
            {
            Pwm_Channel = 0;
            Pwm_Channel_Brightness = 4095;
            if ((StagesLeft >= 1) and (ModuleCount == PWMModules -1))
                      {
                      Calc_Num_Stages_per_Module = StagesLeft;
                      }
            else
                      {
            Calc_Num_Stages_per_Module = Num_Stages_per_Module;
                      }
            while (Pwm_Channel < Calc_Num_Stages_per_Module +1)
              {
              Wire.beginTransmission( PWM_Module_Base_Addr + ModuleCount);
              Wire.write(Pwm_Channel * 4 +8);    // Wähle PWM_Channel_0_OFF_L register
              Wire.write((byte)Pwm_Channel_Brightness & 0xFF);       // Wert für o.g. Register
              Wire.endTransmission();
              Wire.beginTransmission(PWM_Module_Base_Addr + ModuleCount);
              Wire.write(Pwm_Channel * 4 +9);   // Wähle PWM_Channel_0_OFF_H register
              Wire.write((Pwm_Channel_Brightness >> 8));           // Wert für o.g. Register
              Wire.endTransmission();
             if (Pwm_Channel_Brightness > 0)
               {
               Pwm_Channel_Brightness = Pwm_Channel_Brightness - MyConfig.Delay_Stages_OFF;
```

```
                if (Pwm_Channel_Brightness < 0) {Pwm_Channel_Brightness = 0;}
                } else if ( Pwm_Channel < Num_Stages_per_Module +1)
                {
                  Pwm_Channel_Brightness = 4095;
                  delay(MyConfig.delay_per_Stage_in_ms);
                  Pwm_Channel++;
                }
              }
            }
}

void Up_to_DOWN_OFF()
{
#ifdef DEBUG
Serial.println(F("Up_to_DOWN_OFF"));
#endif
byte Calc_Num_Stages_per_Module = Num_Stages_per_Module;
int ModuleCount = PWMModules - 1;
while (ModuleCount >= 0)
            {
            Pwm_Channel_Brightness = 4095;
            if ((StagesLeft >= 1) and (ModuleCount == PWMModules -1))
                        {
                        Calc_Num_Stages_per_Module = StagesLeft;
                        }
            else
                        {
                         Calc_Num_Stages_per_Module = Num_Stages_per_Module;
                        }
            Pwm_Channel = Calc_Num_Stages_per_Module;
            while (Pwm_Channel > -1)
              {
              Wire.beginTransmission(PWM_Module_Base_Addr + ModuleCount);
              Wire.write(Pwm_Channel * 4 +8);    // Wähle PWM_Channel_0_OFF_L register
              Wire.write((byte)Pwm_Channel_Brightness & 0xFF);       // Wert für o.g. Register
              Wire.endTransmission();
              Wire.beginTransmission(PWM_Module_Base_Addr + ModuleCount);
              Wire.write(Pwm_Channel * 4 +9);   // Wähle PWM_Channel_0_OFF_H register
              Wire.write((Pwm_Channel_Brightness >> 8));           // Wert für o.g. Register
              Wire.endTransmission();
              if (Pwm_Channel_Brightness > 0)
               {
               Pwm_Channel_Brightness = Pwm_Channel_Brightness - MyConfig.Delay_Stages_OFF;
               if (Pwm_Channel_Brightness < 0) {Pwm_Channel_Brightness = 0;}
               } else if ( Pwm_Channel >= 0)
               {
                 Pwm_Channel_Brightness =  4095;
                 delay(MyConfig.delay_per_Stage_in_ms);
                 Pwm_Channel--;
                if ( Pwm_Channel < 0)
                 {
                   Pwm_Channel =0;
                   break;
```

```
                    }
                  }
                 }
            ModuleCount = ModuleCount -1;
            }
}

void Stages_Light_Control ()
{
 if ((Motion_Trigger_Down_to_Up) and !(On_Delay))
  {
  DLightCntrl = DayLightStatus();
  if (DLightCntrl)
    {
    Seconds24 = 0;
    On_Delay = true;
    Down_to_Up_ON();
    } else { Motion_Trigger_Down_to_Up = false; }
  }
if ((On_Delay) and (Seconds24 > MyConfig.Delay_ON_to_OFF) and (Motion_Trigger_Down_to_Up)
)
  {
   Down_to_Up_OFF();
   Motion_Trigger_Down_to_Up = false;
   On_Delay = false;
   Seconds24 = 0;
  }
if ((Motion_Trigger_Up_to_Down) and !(On_Delay))
  {
  DLightCntrl = DayLightStatus();
  if (DLightCntrl)
    {
    Seconds24 = 0;
    On_Delay = true;
    Up_to_DOWN_ON();
    } else { Motion_Trigger_Up_to_Down = false; }
  }
if ((On_Delay) and (Seconds24 > MyConfig.Delay_ON_to_OFF) and
(Motion_Trigger_Up_to_Down))
  {
   Up_to_DOWN_OFF();
   Motion_Trigger_Up_to_Down = false;
   On_Delay = false;
   Seconds24 = 0;
  }
}

//Serial Command Interpreter Functions ------------------------------

void ClearCBuffer ()
{
 for (byte a= 0; MaxInputBufferSize -1;a++)
  Cbuffer[a] = 0;
```

```
}

boolean CheckforserialEvent()
{
  while (Serial.available()) {
    // get the new byte:
    TBuffer = Serial.read();
    if (TBuffer > 9 && TBuffer < 14)
      {
       Cbuffer[Ccount] = 0;
       TBuffer =0;
       Serial.print(char(13));
       Serial.flush();
       Serial.println("");
       Sbuffer = "";
       value = 0;
       EnterInput = true;
      return true;
    } else if (TBuffer > 47 && TBuffer <58 )
     {
      if ( Ccount < MaxInputBufferSize)
        {
          Cbuffer[Ccount] = TBuffer;
          Ccount++;
        } else {Serial.print("#"); }
     //Number Input detected
     NumberInput = true;
     }
     else if (TBuffer > 64 && TBuffer < 123 )
     {
      if ( Ccount < MaxInputBufferSize)
        {
          Cbuffer[Ccount] = TBuffer;
          Ccount++;
          Serial.print(char(TBuffer));
          Serial.flush();
        }
     //Character Char Input detected
     StrInput = true;
    }
   else if ( (TBuffer == 127 )  |  (TBuffer == 8 ) )
     {
       if ( Ccount > 0)
        {
          Ccount--;
          Cbuffer[Ccount] = 0;
          Serial.print("-");
          Serial.flush();
        }
     }
   else
     {
      if ( Ccount < MaxInputBufferSize)
```

```
        {
          Cbuffer[Ccount] = TBuffer;
          Ccount++;
          Serial.print(char(TBuffer));
          Serial.flush();
     //Data Input detected
     DataInput = true;
     }
    return false;
    }
  return false;
  }
}

byte SerInputHandler()
{
byte result = 0;
int c;
int d;
int a;
int b;
result = 0;
if (CheckforserialEvent())
   {
    if ((NumberInput) and not (DataInput)and not (StrInput))     //Numbers only
      {
      Sbuffer = "";
      value = 0;
      StrInput = false;
      NumberInput = false;
      DataInput = false;
      EnterInput = false;
      a = 0;
      b = 0;
      c = 0;
      d = 0;
      Sbuffer = Cbuffer; // Zahl wird AUCH ! in SBUFFER übernommen, falls benötigt.
      if (Ccount == 1) { value  = Cbuffer[0]- 48 ; }
      if (Ccount == 2) {
        a = Cbuffer[0] - 48 ;
        a = a * 10;
        b = Cbuffer[1] - 48 ;
        value = a + b;
        }
      if (Ccount == 3) {
        a = Cbuffer[0] - 48 ;
        a = a * 100;
        b = Cbuffer[1] - 48 ;
        b = b * 10;
        c = Cbuffer[2] - 48 ;
        value = a + b + c;
        }
      if (Ccount == 4) {
```

```
    a = Cbuffer[0] - 48 ;
    a = a * 1000;
    b = Cbuffer[1] - 48 ;
    b = b * 100;
    c = Cbuffer[2] - 48 ;
    c = c * 10;
    d = Cbuffer[3] - 48 ;
    value = a + b + c + d;
    }
  if (Ccount >= 5)
    {
     Sbuffer = "";
     value = 0;
     Sbuffer = Cbuffer;
     ClearCBuffer;
     result = 2;
    } else
    {
     ClearCBuffer;
     Ccount = 0;
     result = 1;                              //Number Returncode
     NumberInput = false;
     StrInput = false;
     DataInput = false;
     EnterInput = false;
     Ccount = 0;
     return result;
     }
  }
 if ((StrInput) and not (DataInput))              //String Input only
  {
  Sbuffer = "";
  Sbuffer = Cbuffer;
  value = 0;
  StrInput = false;
  NumberInput = false;
  DataInput = false;
  EnterInput = false;
  Ccount = 0;
  ClearCBuffer;
  result = 2;                               //Number Returncode
  }
 if (DataInput) {
  Sbuffer = "";
  Sbuffer = Cbuffer;
  value = 0;
  StrInput = false;
  NumberInput = false;
  DataInput = false;
  EnterInput = false;
  Ccount = 0;
  ClearCBuffer;
  result = 3;                              //Number Returncode
```

```
    }
    if ((EnterInput) and not (StrInput) and not (NumberInput) and not (DataInput))
    {
    Sbuffer = "";
    value = 0;
    Ccount = 0;
    ClearCBuffer;
    result = 4;                              //Number Returncode
    }
  NumberInput = false;
  StrInput = false;
  DataInput = false;
  EnterInput = false;
  Ccount = 0;
  return result;
 }
return result;
  //End CheckforSerialEvent
}

void SerialcommandProcessor()
{
int a;
Inptype = 0;
Inptype = SerInputHandler();
// 0 keine Rückgabe
// 1 Nummer
// 2 String
// 3 Data
if (Inptype > 0)
  {
  MenueSelection = 0;
  if ((MnuState < 2) && (Inptype == 2)) {Sbuffer.toUpperCase(); } // For Easy Entering Commands
  if ((Sbuffer == "D") && (MnuState == 0) && (Inptype == 2))   { MenueSelection = 1;}
  if ((Sbuffer == "O")&& (MnuState == 0) && (Inptype == 2))      { MenueSelection = 2;}
  if ((Sbuffer == "T") && (MnuState == 0) && (Inptype == 2))      { MenueSelection = 3;}
  if ((Sbuffer == "B") && (MnuState == 0) && (Inptype == 2))      { MenueSelection = 4;}
  if ((Sbuffer == "N") && (MnuState == 0) && (Inptype == 2))   { MenueSelection = 5;}
  if ((Sbuffer == "F")&& (MnuState == 0) && (Inptype == 2))      { MenueSelection = 6;}

  if ((MnuState == 2) && (Inptype == 1))                  { MenueSelection = 8;}
  if ((MnuState == 3) && (Inptype == 1))                  { MenueSelection = 9;}
  if ((MnuState == 4) && (Inptype == 1))                  { MenueSelection = 10;}
  if ((MnuState == 5) && (Inptype == 1))                  { MenueSelection = 11;}
  if ((MnuState == 6) && (Inptype == 1))                  { MenueSelection = 12;}
  if ((MnuState == 7) && (Inptype == 1))                  { MenueSelection = 13;}

  if (MnuState == 10)                              { MenueSelection = 21;} // Time Set
  if (MnuState == 11)                              { MenueSelection = 24;} // Time Set
  if (MnuState == 12)                              { MenueSelection = 25;} // Time Set
  if (MnuState == 13)                              { MenueSelection = 27;} // Background Set
  if (MnuState == 14)                              { MenueSelection = 29;} // ClockFace Set
  switch (MenueSelection)
```

```
  {
    case 1:
    {
    Serial.println("Delay ON to OFF: (1-65000)");
    MnuState = 2;
    value = 0;
    Sbuffer = "";
    break;
    }
    case 2:
    {
    Serial.println("Overall Stages: (1-992)");
    MnuState = 3;
    value = 0;
    Sbuffer = "";
    break;
    }
    case 3:
    {
    Serial.println("Delay per Stage in ms: (1-65000)");
    MnuState = 4;
    value = 0;
    Sbuffer = "";
    break;
    }
    case 4:
    {
    Serial.println("DayLight Brightness Border: (0-65000)");
    MnuState = 5;
    value = 0;
    Sbuffer = "";
    break;
    }
    case 5:
    {
    Serial.println("Delay Stages ON: (1-254)");
    MnuState = 6;
    value = 0;
    Sbuffer = "";
    break;
    }
    case 6:
    {
    Serial.println("Delay Stages OFF: (1-254)");
    MnuState = 7;
    value = 0;
    Sbuffer = "";
    break;
    }
    case 8:
    {
    MyConfig.Delay_ON_to_OFF = value;
    saveEEPROM_Config();
```

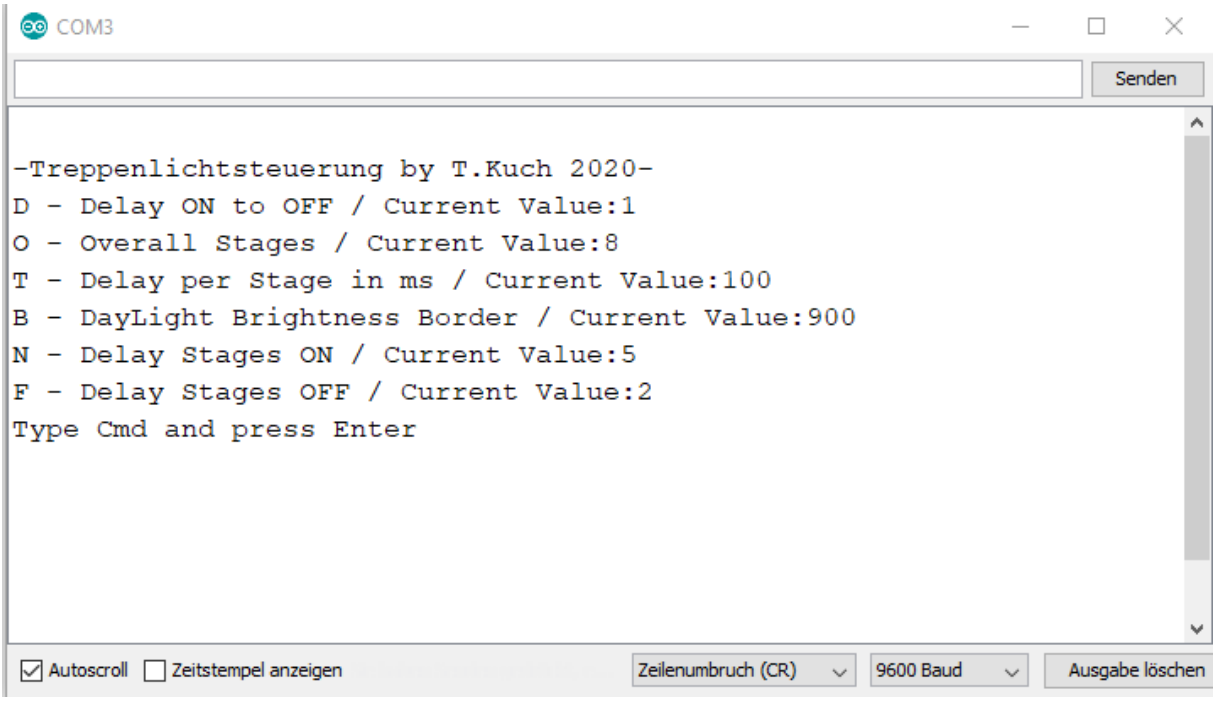
```
Serial.print(F("Delay_ON_to_OFF set to:"));
Serial.println(MyConfig.Delay_ON_to_OFF);
MnuState = 0;
Sbuffer = "";
value = 0;
break;
}
case 9:
{
MyConfig.Overall_Stages = value;
saveEEPROM_Config();
Serial.print(F("Overall Stages set to:"));
Serial.println(MyConfig.Overall_Stages);
MnuState = 0;
Sbuffer = "";
value = 0;
break;
}
case 10:
{
MyConfig.delay_per_Stage_in_ms = value;
saveEEPROM_Config();
Serial.print(F("Delay per Stage in ms set to:"));
Serial.println(MyConfig.delay_per_Stage_in_ms);
MnuState = 0;
Sbuffer = "";
value = 0;
break;
}
case 11:
{
MyConfig.DayLight_Brightness_Border = value;
saveEEPROM_Config();
Serial.print(F("DayLight Brightness Border set to:"));
Serial.println(MyConfig.DayLight_Brightness_Border);
MnuState = 0;
Sbuffer = "";
value = 0;
break;
}
case 12:
{
MyConfig.Delay_Stages_ON = value;
saveEEPROM_Config();
Serial.print(F("Delay Stages ON set to:"));
Serial.println(MyConfig.Delay_Stages_ON);
MnuState = 0;
Sbuffer = "";
value = 0;
break;
}
case 13:
{
```

```
      MyConfig.Delay_Stages_OFF = value;
      saveEEPROM_Config();
      Serial.print(F("Delay Stages OFF set to:"));
      Serial.println(MyConfig.Delay_Stages_OFF);
      MnuState = 0;
      Sbuffer = "";
      value = 0;
      break;
      }

      default:
      {
        MnuState = 0;
       Serial.println(F("-Treppenlichtsteuerung -"));
       Serial.print(F("D - Delay ON to OFF / Current Value:"));
       Serial.println(MyConfig.Delay_ON_to_OFF);
       Serial.print(F("O - Overall Stages / Current Value:"));
       Serial.println(MyConfig.Overall_Stages);
       Serial.print(F("T - Delay per Stage in ms / Current Value:"));
       Serial.println(MyConfig.delay_per_Stage_in_ms);
       Serial.print(F("B - DayLight Brightness Border / Current Value:"));
       Serial.println(MyConfig.DayLight_Brightness_Border );
       Serial.print(F("N - Delay Stages ON / Current Value:"));
       Serial.println(MyConfig.Delay_Stages_ON);
       Serial.print(F("F - Delay Stages OFF / Current Value:"));
       Serial.println(MyConfig.Delay_Stages_OFF);
       Serial.println(F("Type Cmd and press Enter"));
       Serial.flush();
       MnuState = 0;
       value = 0;
       Sbuffer = "";
      }
    }
   } // Eingabe erkannt
}

void loop()
{
Stages_Light_Control();
SerialcommandProcessor();
}
```

Nachdem der Codehochgeladen wurde, können wir uns mit 9600 Baud auf die serielle Schnisstelle verbinden. Nach einem Enter (und inaktivem! Treppenlicht) erscheint folgendes Konfigurationsmenü:



Paramenter	**Erklärung**
Delay ON to OFF	Zeit in SEKUNDEN, die die Treppenbeleuchtung vollständig eingeschaltet bleibt
Overall Stages	Anzahl an Treppenstufen der Treppe
Delay per Stage	Zeit in MILLISEKUNDEN, die gewartet wird, bis die nächste Treppe angesteuert wird.
Daylight Brightness Border	Helligkeit, in der die Treppenbeleuchtung inaktiv wird. Höherer Wertz -> höhere Helligkeit
Delay Stages ON	rel. Fadingzeit beim EINSCHALTEN der Treppen. Höherer Wert - > kleinere Zeit
Delay Stages OFF	rel. Fadingzeit beim AUSSCHALTEN der Treppen. Höherer Wert - > kleinere Zeit

Ich wünsche viel Spaß beim Nachbau. Wie immer findet Ihr auch alle vorherigen Projekte unter der GitHub Seite https://github.com/kuchto