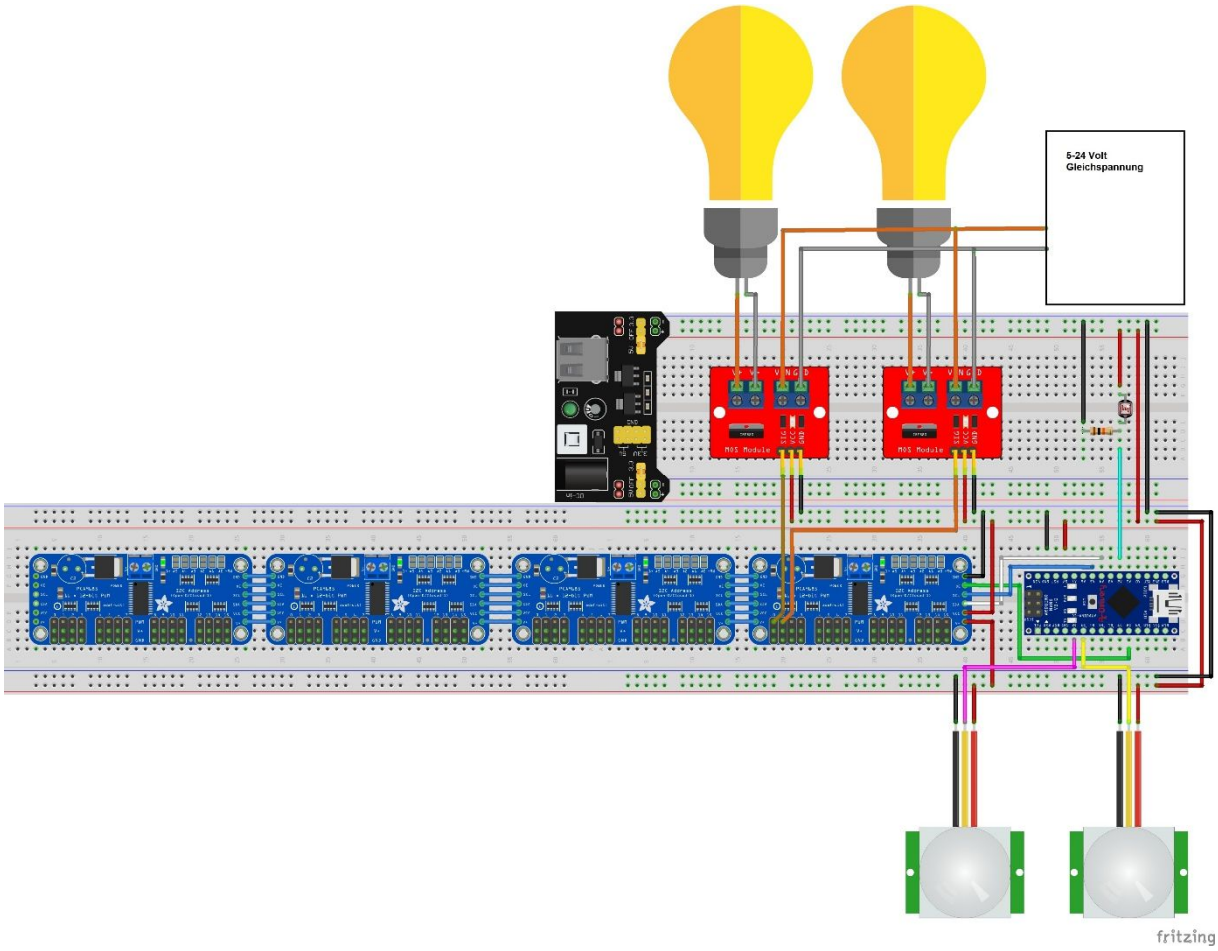
Eine elegante automatische Treppenbeleuchtung (Teil4)

Hallo und Willkommen zu dem vorletzten der Teil der Reihe „elegante automatische Treppenbeleuchtung“. Heute erweitern wir unsere Steuerung um einen lichtempfindlichen Wiederstand, der als Helligkeitssensor fungieren soll. Ab einem im Code mit dem Parameter „DayLight_Brightness_Border“ einstellbaren Helligkeitswert, wobei eine höhere Zahl eine höhere Helligkeit symbolisiert, wird die Treppenautomatik ab diesem Wert deaktiviert, sodass nur, wenn es dunkel ist, die Treppe beleuchtet wird. Durch die Verwendung eines LDRs anstatt eines I2C Sensors, wird das ganze etwas unempfindlicher gegen externe Störungen. Als weitere kleine in Verbesserung kann in dieser Version, die Fadingzeit zwischen zwei Stufen beim Ein und ausschalten durch die Parameter „Delay_Stages_ON“ und „Delay_Stages_OFF“ nun getrennt voneinander eingestellt werden. Respektable Ergebnisse lassen sich zum Beispiel dadurch erzielen, indem der Parameter für Ausschalten der einzelnen Treppenstufen größer gewählt wird als der für das einschalten.

Im nachfolgenden Bild ist erkennbar, wie der LDR und der Vorwiederstand verschaltet werden müssen



Für unseren heutigen Teil des Projektes benötigen wir:

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

Alle Hinweise aus den vorherigen Teilen gelten auch im heutigen Teil.

Nach eigenen Anpassungen und des Hinzufügens der LDR‘s kann der Code hochgeladen werden:

```
#include <Wire.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

// 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;


// 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;

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 = Overall_Stages / 16;
  StagesLeft = (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);
   }
  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"));
}

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

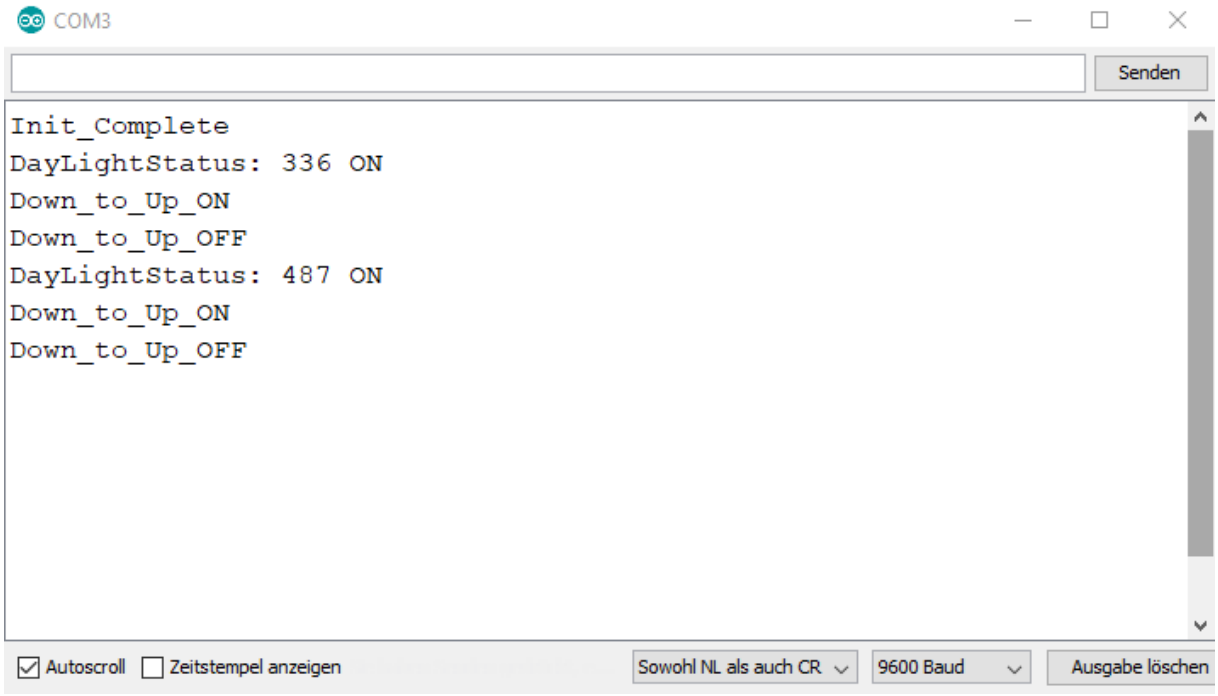
```
              if (Pwm_Channel_Brightness < 4095)
               {
               Pwm_Channel_Brightness = Pwm_Channel_Brightness + Delay_Stages_ON;
               if (Pwm_Channel_Brightness > 4095) {Pwm_Channel_Brightness = 4095;}
               } else if ( Pwm_Channel < Num_Stages_per_Module +1)
               {
                 Pwm_Channel_Brightness = 0;
                      delay(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 + Delay_Stages_ON;
             if (Pwm_Channel_Brightness > 4095) {Pwm_Channel_Brightness = 4095;}
             } else if ( Pwm_Channel >= 0)
            {
              Pwm_Channel_Brightness = 0;
             delay(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 - 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(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 - Delay_Stages_OFF;
             if (Pwm_Channel_Brightness < 0) {Pwm_Channel_Brightness = 0;}
             } else if ( Pwm_Channel >= 0)
             {
               Pwm_Channel_Brightness =  4095;
               delay(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 > 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 > 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;
  }
}

void loop()
{
Stages_Light_Control ();

}
```

Zu Debugzwecken steht eine serielle 9600 Baud Verbindung zur Verfügung an der einige Informationen zum aktuellen Status ausgegeben werden:



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