Eine elegante automatische Treppenbeleuchtung (Teil2)

Willkommen zum zweiten Teil unserer Reihe " elegante Treppenlichtsteuerung". Wie immer in den Projektreihen geht es in den Folgeteilen um eine Verbesserung oder Erweiterungen der Funktion. Im heutigen Teil verbessern wir zunächst die Funktion.

Grundsätzlich geht unsere Treppe schon jedes langsam von unten nach oben an, wenn man diese von oben nach unten oder umgekehrt, von unten nach oben betritt. Schöner währe es jedoch, wenn unsere Treppe sich genau in der Richtung als Lauflicht einschaltet, in der wir die Treppe betreten und danach wieder langsam ausschaltet.

Genau um diese Erweiterung soll es im heutigen Teil der Reihe gehen. Das Treppenlicht folgt unseren Schritten, sobald wir die Treppe betreten, unabhängig von der Richtung.

Die Parameter und die Schaltung können aus dem ersten Teil der Reihe übernommen werden. Es handelt sich um eine reine Softwareerweiterung.

```
#define Num_Stages  15
#define Delay_Stages  10
#define Delay_ON_to_OFF  5
```

Num_Stages	Definiert die Anzahl der zu beleuchtenden Treppen (maximal 16, von 0 anzählend zu beginnen. Maximalwert: 15)
Delay_Stages	Fade Zeitraum für jede Treppenstufe -> je kleiner der Wert desto größer der Zeitraum, desto langsamer.
Delay_ON_to_OFF	Zeitraum der vergeht, indem die Treppe im Status „an“ verbleibt.

Nachdem die Werte den eigenen Vorlieben angepasst wurden, kann der erweiterte Code auf den Arduino hochgeladen werden:

```
// 2019 Tobias Kuch GPL 3.0
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
                  //wird ein Lesevorgang auswählt, während eine logische 0 eine Schreiboperation
auswählt.
#define OE_Pin  8         // Pin für Output Enable
#define CPU_LED_Pin 13
#define PIRA_Pin 2
#define PIRB_Pin 3

#define Num_Stages  15
#define Delay_Stages  5
#define Delay_ON_to_OFF  30  // Minimum Delay_ON_to_OFF in Seconds
```

```
#define delay_per_Stage_in_ms 200

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;

// interrupt Control
byte A60telSeconds24 = 0;
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)
{
  pinMode(OE_Pin,OUTPUT);
  pinMode(CPU_LED_Pin,OUTPUT);
  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);
  Serial.begin(9600);
  Wire.begin(); // Initalisiere I2C Bus A4 (SDA), A5 (SCL)
  Init_PWM_Module(PWM_Module_Base_Addr);
  Init_PWM_Outputs(PWM_Module_Base_Addr);
  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();
}

void Down_to_Up_ON()
{
 Pwm_Channel = 0;
 Pwm_Channel_Brightness = 0;
 while (Pwm_Channel < Num_Stages +1)
   {
   Wire.beginTransmission( PWM_Module_Base_Addr);
   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);
   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;
    if (Pwm_Channel_Brightness > 4095) {Pwm_Channel_Brightness = 4095;}
    } else if ( Pwm_Channel < Num_Stages +1)
    {
      Pwm_Channel_Brightness = 0;
      delay(delay_per_Stage_in_ms);
```

```
      Pwm_Channel++;
     }

   }
}

void Down_to_Up_OFF()
{
 Pwm_Channel = 0;
 Pwm_Channel_Brightness = 4095;
 while (Pwm_Channel < Num_Stages +1)
   {
   Wire.beginTransmission( PWM_Module_Base_Addr);
   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);
   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;
    if (Pwm_Channel_Brightness < 0) {Pwm_Channel_Brightness = 0;}
    } else if ( Pwm_Channel < Num_Stages +1)
    {
      Pwm_Channel_Brightness = 4095;
      delay(delay_per_Stage_in_ms);
      Pwm_Channel++;
    }
   }
}

void Up_to_DOWN_ON()
{
 Pwm_Channel = Num_Stages;
 Pwm_Channel_Brightness = 0;
 while (Pwm_Channel > -1)
   {
   Wire.beginTransmission( PWM_Module_Base_Addr);
   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);
   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;
    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;
       }
    }
   }
}



void Up_to_DOWN_OFF()
{
 Pwm_Channel = Num_Stages;
 Pwm_Channel_Brightness = 4095;
 while (Pwm_Channel > -1)
   {
   Wire.beginTransmission(PWM_Module_Base_Addr);
   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);
   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;
    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;
       }
    }
   }
}




void loop()
{
 if ((Motion_Trigger_Down_to_Up) and !(On_Delay) )
```

```
  {
  Seconds24 = 0;
  On_Delay = true;
  Down_to_Up_ON();

  }
 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) )
  {
  Seconds24 = 0;
  On_Delay = true;
  Up_to_DOWN_ON();
  }

 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;

  }
}
```

Ich wünsche viel Spaß beim Nachbauen dieses Projektes und bis zum nächsten Teil.