Eine elegante automatische Treppenbeleuchtung (Teil3)

Hallo und Willkommen zu einer Fortsetzung der Reihe elegante Treppenbeleuchtung. Heute erweitern wie die maximale Treppenstufenanzahl, die unser Controller ansteuern kann von gerade einmal 16 auf Fulminate 992 Treppenstufen!

Somit sollten auch Besitzer einer längeren Stufentreppe problemlos die Steuerung für Ihre Zwecke entsprechend Ihren Bedürfnissen und Wünschen anpassen können.

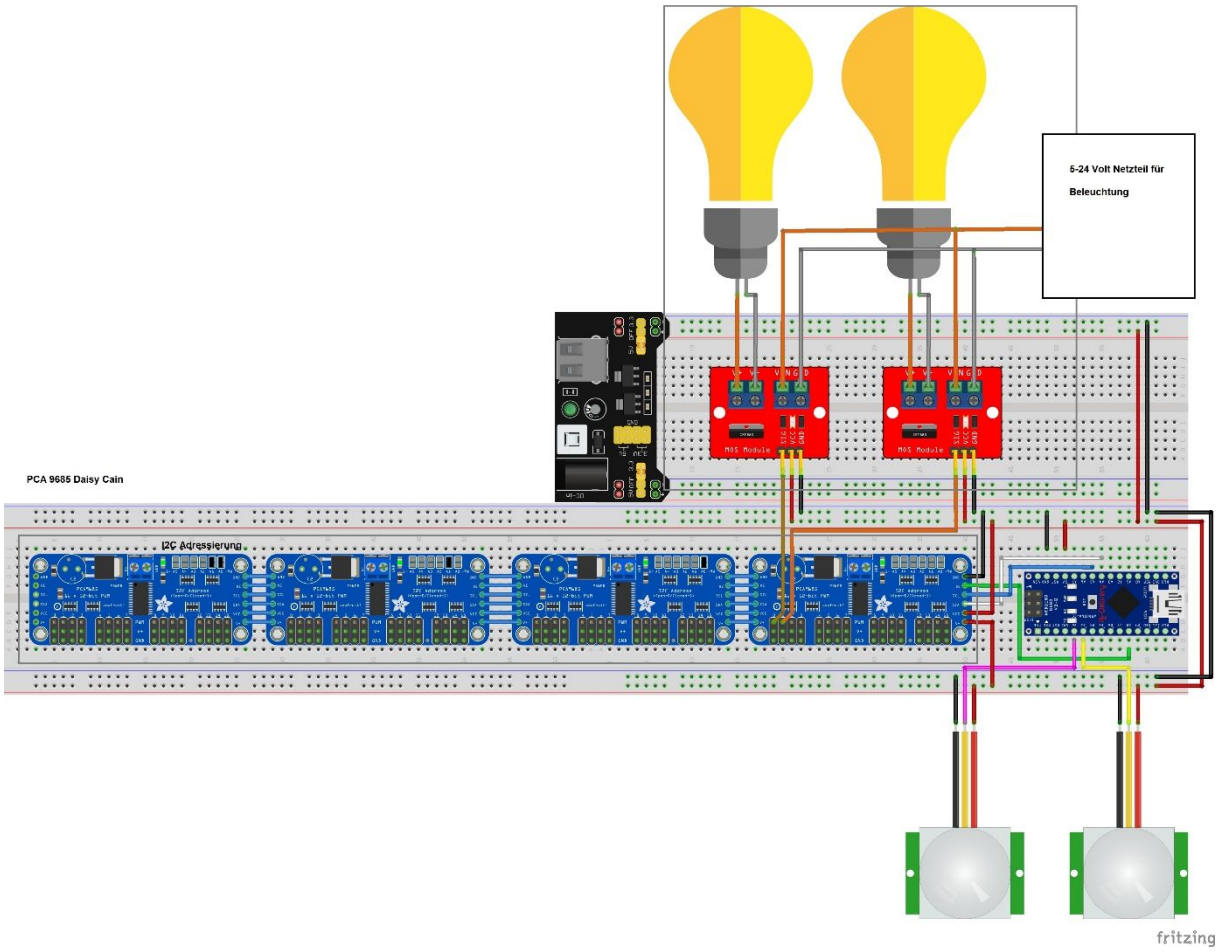
Diese Anpassung erfolgt dabei einfach und bequem durch die Anpassung der Programmvariablen "Overall_Stages" in die gewünschte Treppenzahl von 1 bis 992.

Nun, der eine oder andere mag sich fragen, wie ich diese große Anzahl an PWM Ausgängen bei gerade einmal 20 verfügbaren Ausgängen eines Arduino Nanos erreiche. Das Geheimnis liegt in den 6 Adressbits des PCA9685. Jeder PCA9685 kann 16 PWM Kanäle zur Verfügung stellen. Das Daisy-Chain-Layout des Boards verhindert zu dem Kabelchaos und ermöglicht die komfortable Verbindung der bis zu 62 Controller.

Durch die binäre Adressierung der Boards kann eine von insgesamt 62 Adressen im Bereich von 40h bis 7Eh einem Modul aufwärts zählend, zugeordnet werden. Dies ergibt dann bei 16 Ausgängen pro Modul die maximal möglichen 992 Ausgänge.

Der Code ist von mir so geschrieben worden, dass die Adressierung der Module vom Benutzer abstrahiert wird und automatisch berechnet wird. Es ist lediglich die Gesamtsumme der Treppen in einer Variablen angegeben werden und eine ansprechende Anzahl an Modulen als Daisy Chain an den I2C Bus angehängt werden. Dabei muss bei den Modulen auf eine aufsteigende Adressierung der Module geachtet werden.

Im nachfolgenden Bild ist erkennbar, wie die Module verschaltet und adressiert werden müssen:



Für unser heutiges Projekt 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

Bitte beachtet, dass die Schaltung nicht mit Vollbestückung getestet wurde, sondern nur mit 2 Modulen vom Typ PCA9685. Eine Vollbestückung mit 62 Modulen ist daher hier nur **theoretisch** beschrieben. In der Praxis sind maximale I2C Buslängen, Störungssicherheit, saubere Signalübertragung, ausreichend dimensionierte Netzteile, Kabel, sichere Verbindungen und mehr zu berücksichtigen. Alle Hinweise dazu aus den vorherigen Teilen gelten insbesondere auch im heutigen Teil.

Hier ist der aktualisierte Code für den heutigen Teil:

```
// Code by Tobias Kuch 2019 mailto:tobias.kuch@googlemail.com
// Code under 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 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
#define PIRA_Pin 2
#define PIRB_Pin 3
#define Num_Stages_per_Module 16
#define Delay_Stages  5
#define Delay_ON_to_OFF  2  // Minimum Delay_ON_to_OFF in Seconds


int Overall_Stages =  16; // maximale Stufenanzahl: 31 x 16 = 496
int 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;
byte PWMModules = 0;
byte StagesLeft = 0;
// 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(115200);
   pinMode(PIRA_Pin,INPUT);
   pinMode(PIRB_Pin,INPUT);
   Serial.begin(9600);
   while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB
   }
   PWMModules = Overall_Stages / 16;
   StagesLeft = Overall_Stages % 16;
   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();
}

void Down_to_Up_ON()
{
Serial.println("Down_to_Up_ON");
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;
                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()
{
Serial.println("Up_to_DOWN_ON ");
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;
```

```
                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()
{
Serial.println("Down_to_Up_OFF");
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;
                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()
{
Serial.println("Up_to_DOWN_OFF");
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;
                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 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 Nachbau und bis zum nächsten Teil der Reihe. Wie immer findet Ihr auch alle vorherigen Projekte unter der GitHub Seite https://github.com/kuchto